UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Analizador sintáctico del español accesible online basado en gramática de dependencias

Javier Juárez Rodríguez
Tutor: Pablo Alfonso Haya Coll
Ponente: Germán Montoro Manrique

Marzo 2019

Analizador sintáctico del español accesible online basado en gramática de dependencias

AUTOR: Javier Juárez Rodríguez
TUTOR: Pablo Alfonso Haya Coll

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Marzo 2019

Resumen (castellano)

Este Trabajo Fin de Grado tiene como objetivo la creación de una aplicación online que permita analizar sintácticamente oraciones en el lenguaje español. Dicha aplicación utilizará analizadores sintácticos basados en gramática de dependencias. Este Trabajo de Fin de Grado es una idea conjunta de los Departamentos de Lingüística e Informática de la Universidad Autónoma de Madrid.

En el documento se explica el porqué de la creación de ciertas herramientas de las que carece el software del analizador sintáctico, como por ejemplo, la edición del análisis en caso de que éste haya sido realizado erróneamente.

La finalidad de este proyecto es el desarrollo de un algoritmo, realizado en lenguaje Java, que permitirá analizar oraciones escritas directamente en la aplicación, desde un fichero o una URL, y editar los resultados que se observen erróneos.

La aplicación en la que se basa este proyecto, consiste en que el usuario puede introducir una oración y la aplicación la analizará utilizando uno de los dos, o los dos, analizadores que se han usado: SpaCy, un potente analizador sintáctico programado en Python, y UDPipe, un analizador nacido de un proyecto de investigación y con un binding para poder usarlo en Java. Ambos son capaces de trabajar con el formato CoNLL-U, que será el que utilicemos en nuestra aplicación.

Abstract (English)

This Bachelor Thesis aims to create an online application that will syntactically analyse phrases in the Spanish language. This application will use syntactic analysers based on a dependency grammar model. This Bachelor Thesis is an idea of the Linguistics and Computing Departments of the Universidad Autónoma de Madrid.

This document explain why to create certain tools that are not present in the syntactic analyser software. For example, edit the resultant analysis in case it is wrong.

The goal of this project is developing an algorithm, programmed in the Java programming language, which will allow to analyse phrases written in the application, taken from a text file or from an URL, and edit the results if they are wrongly analysed.

The application on which this project is based, is developed so the user can introduce a phrase and the application will analyse it using one or both of the analysers that have been used: SpaCy, a powerful syntactic analyser developed in the Python programming language, and UDPipe an analyser born from an investigation project and that has a binding for using it in the Java programming language. Both analysers use the CoNLL-U format, which will be the one we will use in our application.

Palabras clave (castellano)

Analizador, root, dependencias, treebank, corpus

Keywords (inglés)

Parser, root, dependencies, treebank, corpus

Agradecimientos

A mi familia, en especial a mis padres, por todo el apoyo que me han dado durante todos los años de carrera.

A mi tutor Pablo Haya y a Antonio Moreno Sandoval, por su trabajo y ayuda durante la realización de este proyecto.

A todos mis compañeros de la carrera, en especial a los amigos que conocí al entrar en dicha carrera.

A María, por el apoyo, la paciencia y el ánimo en los momentos buenos, y sobre todo en los malos.

Muchas gracias a todos.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

El Trabajo de Fin de Grado "Analizador sintáctico del español accesible online basado en gramática de dependencias" nace de una idea conjunta del Departamento de Lingüística y el Departamento de Informática de la Universidad Autónoma de Madrid, que persigue explorar las posibilidades que presentan las gramáticas de dependencias aplicadas al Procesamiento del Lenguaje Natural.

La sintaxis es, probablemente, la rama más importante del análisis lingüístico, aunque está fuertemente relacionado con el análisis morfológico. De hecho, a veces se le denomina análisis morfosintáctico. La sintaxis se puede definir como la parte de la lingüística que estudia la relación entre las palabras que forman las oraciones, así como sus funciones dentro de dicha oración. Por otro lado, la morfología estudia la clasificación y el funcionamiento de las palabras en la oración, así como sus derivaciones.

En este trabajo, se utilizará en particular el análisis sintáctico basado en gramática de dependencias. La gramática de dependencias consiste en establecer relaciones de dependencia entre las palabras de una oración. Se considera que una palabra es dependiente de otra si esta última es necesaria para que exista la primera en dicha oración. Se usa este tipo de análisis porque es mucho más efectivo a la hora de analizar frases en el lenguaje español entre otros. Anteriormente se utilizaba el análisis basado en la gramática de constituyentes pero, aunque funcionaba de una manera bastante correcta en inglés, el hecho de que en otros lenguajes como el español, las frases tuvieran una mayor complejidad sintáctica, hizo que se empezara a utilizar el señalado análisis basado en gramática de dependencias.

1.2 Objetivos

El objetivo de este trabajo es el desarrollo de una aplicación online que permita analizar sintácticamente oraciones en español, además de dar la posibilidad de poder editar los resultados obtenidos en caso de que éstos hayan sido analizados erróneamente.

En un principio, se planeaba utilizar el analizador creado por la Universidad de Stanford, llamado Stanford Parser. Pero esta opción se descartó debido a las complicaciones que presentaba a la hora de integrarlo en nuestra aplicación.

En vez de eso, se han utilizado otros dos analizadores: SpaCy, implementado en Python, y UDPipe, un analizador nacido de un proyecto de investigación.

La aplicación ha sido implementada de una forma fácil y usable, de manera que un usuario con conocimientos lingüísticos pueda utilizarla sin experiencia previa en la herramienta. Además, el diseño permite añadir mejoras o nuevas funcionalidades que puedan potenciar la aplicación.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte:** se introducen una serie de conceptos clave y el marco actual en el que está agregado este proyecto.
- **Diseño:** se proporciona un diseño de las herramientas implementadas durante el proyecto.
- **Desarrollo:** se explican las estructuras de las herramientas y los algoritmos más relevantes que las mismas utilizan.
- **Integración, pruebas y resultados:** se muestran las pruebas que certifican las herramientas creadas.

2 Estado del arte

2.1 Procesamiento del lenguaje natural

El Procesamiento del Lenguaje Natural (NLP) se define como un área de investigación y aplicación que explora cómo los ordenadores pueden ser utilizados para entender y manipular lenguaje natural escrito u oral con el objetivo de hacer cosas útiles. [1]

Hay numerosas aplicaciones del NLP, tales como la Inteligencia Artificial, las interfaces de usuario o, como en el caso de este TFG, el análisis morfosintáctico.

Un sistema de NLP primero analizará morfológicamente cada palabra, para después intentar determinar la estructura sintáctica de la oración.

Liddy en 1998, y Feldman en 1999 sugirieron que, para poder entender el lenguaje natural, hay que distinguir entre los siguientes siete niveles interdependientes: [2]

- Nivel fonético, que tiene que ver con la pronunciación.
- Nivel morfológico, que tiene que ver con las partes más pequeñas de las palabras que contienen significado, sufijos y prefijos.
- Nivel léxico, que tiene que ver con el significado léxico de las palabras.
- Nivel sintáctico, que tiene que ver con la gramática y la estructura de las oraciones.
- Nivel semántico, que tiene que ver con el significado de las palabras y las oraciones.
- Nivel del discurso, que tiene que ver con la estructura de los diferentes tipos de textos usando estructuras de documento.
- Nivel pragmático, que tiene que ver con el conocimiento procedente del mundo exterior, es decir, de fuera de los contenidos del documento.

Este campo ha evolucionado mucho a lo largo de los años. El lenguaje de los ordenadores al más bajo nivel se compone únicamente de ceros y unos, lo que se conoce como código binario. La mayoría de la gente no entiende este tipo de lenguaje, por tanto se ha trabajado durante años para desarrollar formas de comunicación con los ordenadores más eficientes y al alcance de todos. Estas formas de comunicación han ido desde las tarjetas perforadas que se usaban en los primeros ordenadores hasta los actuales *Siri*, *Alexa* o *Cortana*, asistentes virtuales capaces de entender nuestras órdenes simplemente con decírselo oralmente. Los avances en el procesamiento del lenguaje natural son los que permiten a estos asistentes entender nuestras órdenes y poder ejecutar un algoritmo u otro en función de ellas.

A pesar de su gran evolución, el procesamiento del lenguaje natural aún tiene un gran camino por recorrer. Hay que tener en cuenta que el lenguaje humano es muy complejo y cambiante. En cada idioma o dialecto hay muchas formas para expresar la misma cosa, además de miles de reglas gramaticales, semánticas y sintácticas que lo convierten en algo muy difícil de interpretar para los ordenadores.

2.2 Formato CoNLL-U

El formato CoNLL-U es el que utilizará nuestra aplicación para mostrar los análisis de las oraciones. Se trata de un formato creado por *Universal Dependencies* como una versión mejorada del formato CoNLL-X [8]. Los analizadores que utilizaremos en nuestra aplicación, SpaCy y UDPipe, son capaces de procesar este formato, entre otros. Este formato muestra, por cada palabra de la oración, una tabla compuesta por 10 campos que contiene el análisis de dicha palabra y las relaciones con el resto de palabras de la oración.

Los campos de la tabla son los siguientes:

- ID: número entero que indica la posición de la cada palabra en la frase. Se reinicia a 1 en cada oración.
- FORM: Forma de la palabra o símbolo de puntuación.
- LEMMA: Lema o tronco de la forma de la palabra.
- UPOS: etiqueta universal que indica la categoría part-of-speech.
- XPOS: otra etiqueta part-of-speech, pero esta vez, específica de cada idioma.
- FEATS: una serie de características morfológicas del inventario universal o de un lenguaje específico.
- HEAD: posición de la palabra con la que se relaciona la palabra actual.
- DEPREL: relación de dependencia universal de la palabra actual con la apuntada por el campo HEAD.
- DEPS: Grafo de dependencias en forma de lista de pares HEAD-DEPREL.
- MISC: cualquier otra información relevante de la palabra.

En la página de *Universal Dependencies* (http://universaldependencies.org/) hay algunos ejemplos de frases en este formato. Vamos a ver el ejemplo con la frase “I have no clue”:

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
1	I	I	PRON	PRP	Case=Nom|Number=Sing|Person=1	2	Nsubj	-	-
2	have	have	VERB	VBP	Number=Sing|Person=1|Tense=Pres	0	Root	-	-
3	no	no	DET	DT	PronType=Neg	4	Det	-	-
4	clue	clue	NOUN	NN	Number=Sing	2	Obj	-	SpaceAfter=No
5	.	.	PUNCT	.	-	2	Punct	-	

Figura 1. Ejemplo grafo formato CoNLL-U

En este ejemplo podemos ver el valor de los campos del formato CoNLL-U en la oración. Hay que destacar especialmente los campos UPOS, HEAD y DEPREL. En el campo UPOS podemos ver la categoría de cada palabra. En el campo HEAD podemos ver la relación de dependencia que tiene la palabra con otra. Por ejemplo, podemos ver que las palabras “I” y “Clue” tienen dependencia de la palabra “Have”. Además, el número 0 que aparece en el HEAD de la palabra “Have” indica que dicha palabra es la raíz de la oración. En el campo DEPREL esto se ve aún más claramente, ya que aparece directamente la palabra ROOT. En el resto de palabras, aparece la relación de dependencia que mantienen con la palabra que indique el campo HEAD.

2.3 Gramática de dependencias

La gramática de dependencias estipula que la estructura sintáctica de una oración consiste en una serie de relaciones entre las partes que conforman la oración. A estas relaciones las llamamos dependencias.

Las dependencias normalmente son unilaterales y se representan en forma de flecha. Dicha flecha lleva el nombre de la relación correspondiente. Por ejemplo: NSUBJ, DET, PREP, etc.

Como explicaremos más adelante, en nuestra aplicación usaremos la herramienta *Graphviz* para dibujar los grafos de los análisis de nuestras oraciones. Estos grafos son un buen ejemplo de gramática de dependencias.

Vamos a ver un ejemplo con la frase *La madre sonreía orgullosa de la actuación de su hijo*.

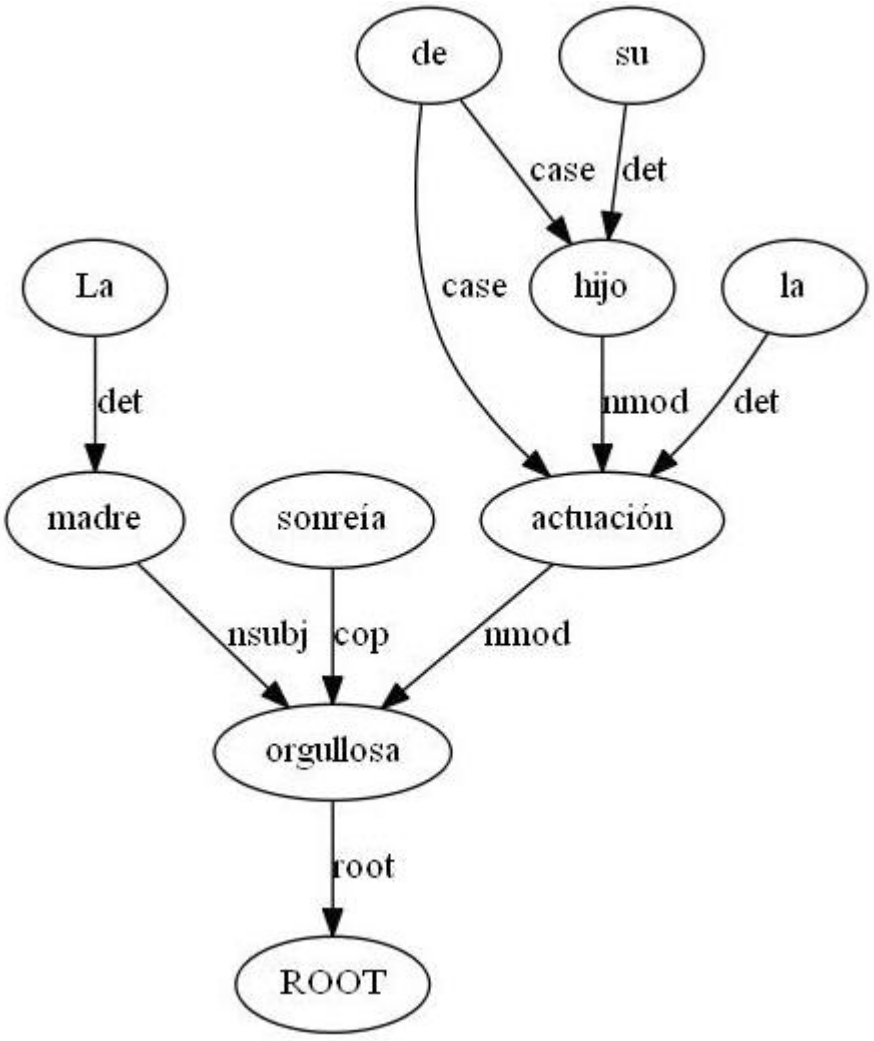


Figura 2. Ejemplo de grafo de análisis de una oración realizado con Graphviz

La teoría de la gramática de dependencias fue propuesta por Lucien Tesnière, un lingüista francés que la expuso por primera vez en 1934 en su artículo *Comment construiré une syntaxe* [4].

Más tarde desarrolló su propuesta en otros libros como *Esquisse d'une syntaxe structurale* (1953) y *Éléments de syntaxe structurale* (1959), en el que se encuentra la versión definitiva de su teoría [5].

2.4 SpaCy

Como hemos comentado, uno de los analizadores que utilizará nuestra aplicación utilizará SpaCy. Se trata de una biblioteca escrita en Python y Cython dirigida al procesamiento de lenguaje natural. Es más eficaz que otras herramientas similares debido a su rapidez y sencillez de uso.

SpaCy contiene modelos para varios idiomas como inglés, alemán, francés, italiano, portugués, holandés y español, que es el que, en este caso, nos ocupa.

Para poder analizar una frase utilizando SpaCy, necesitamos un pequeño programa en Python. A continuación veremos el que hemos utilizado para nuestra aplicación:

```
import spacy
import sys

nlp = spacy.load('es_core_news_sm')
doc = nlp(u'%s' % (sys.argv[1]))

for sent in doc.sents:
        for i, word in enumerate(sent):
                if word.head is word:
                        head_idx = 0
                else:
                        head_idx = word.head.i-sent[0].i+1
                print("%d\t%s\t%s\t%s\t%s\t%s\t%d\t%s\t%s\t%s"%(
                        i+1,
                        word,
                        word.lemma_,
                        word.pos_,
                        '-',
                        word.tag_,
                        head_idx,
                        word.dep_,
                        '-', '-'))
```

Figura 3. Fichero Python utilizado para analizar con SpaCy

Como vemos, se trata de un programa bastante sencillo. Cargamos el modelo *es_core_news_sm*, un modelo del lenguaje español. Después, simplemente con dos bucles, separamos las frases y analizamos cada una de ellas. Podemos apreciar también, que el resultado se devolverá en formato CoNLL-U.

2.5 UDPipe

UDPipe es otro analizador que usaremos en nuestra aplicación. Tiene un binding para utilizarlo en Java, por tanto podremos usarlo sin ningún tipo de problema, tan solo añadiendo la librería a nuestro proyecto.

Es software libre y los distintos modelos pueden usarse libremente para fines no comerciales. Cuenta incluso con una aplicación propia en la que podremos analizar nuestras frases con el modelo que deseemos. Más tarde ahondaremos en ella.

UDPipe puede encontrarse en la siguiente dirección: http://ufal.mff.cuni.cz/udpipe.

UDPipe es un *pipeline* que segmenta, *tokeniza*, etiqueta, *lemmatiza* y realiza análisis de dependencias en oraciones [6]. Tanto en UDPipe como en el formato CoNLL-U, el texto está estructurado en niveles. Ordenados de más grande a más pequeño son:

- Documento
    - Párrafo
        - Oración
            - Token

Los tokens normalmente son palabras, aunque no siempre. Por ejemplo, un token podría estar formado por más de una palabra.

2.6 Stanford Parser

El Stanford Parser es un analizador sintáctico que ha sido creado por la Universidad de Stanford. Se trata de un analizador probabilístico, esto significa que su funcionamiento consiste en seleccionar un análisis para cierta frase en función de la probabilidad de acierto de cada uno.

Esto se consigue *entrenando* el analizador previamente con un determinado número de ejemplos analizados por expertos lingüísticos. A este conjunto de ejemplos se lo denomina *Treebank* o *Corpus*.

Aunque en un principio este analizador fue pensado para el inglés, en los últimos años se están desarrollando modelos para otros idiomas tales como el español, el francés, el chino, el árabe o el alemán.

2.7 Analizadores en línea

A día de hoy, pueden encontrarse varios analizadores sintácticos en la red. Vemos algunos de ellos:

2.7.1 Aplicación UDPipe

Como hemos comentado antes, UDPipe tiene su propia aplicación web para analizar oraciones sintácticamente. Podemos encontrarlo en la siguiente dirección: http://lindat.mff.cuni.cz/services/udpipe/.

En esta aplicación, podremos escribir un texto en el recuadro, o bien subir un fichero con dicho texto. Deberemos seleccionar el modelo que queramos para analizar la oración entre todos los que nos ofrece UDPipe en distintos idiomas. Podemos verlo en la siguiente imagen:

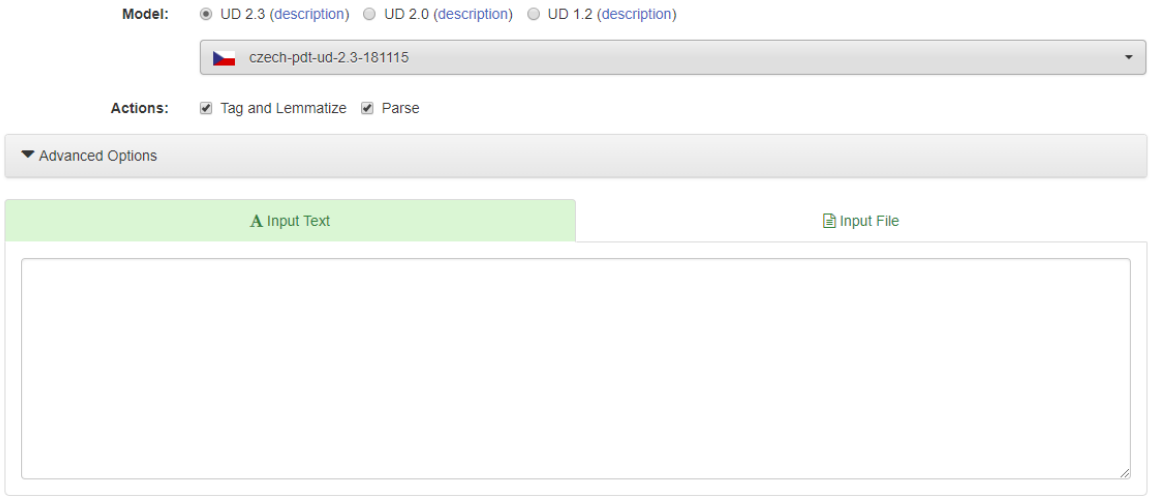


Figura 4. Pantalla principal del analizador de UDPipe

Vamos a probar a analizar la misma oración que en el apartado 2.4: *La madre sonreía orgullosa de la actuación de su hijo.*

El analizador de UDPipe devolverá el resultado en 3 formatos distintos:

- Texto:

```
# newdoc
# newpar
# sent_id = 1
# text = La madre sonreía orgullosa de la actuación de su hijo.
1   La   el   DET DET Definite=Def|Gender=Fem|Number=Sing|PronType=Art   2   det _   _
2   madre   madre   NOUN   NOUN   Gender=Fem|Number=Sing   3   nsubj   _   _
3   sonreía   sonreír   VERB   VERB   Mood=Ind|Number=Sing|Person=3|Tense=Imp|VerbForm=Fin   0   root _   _
4   orgullosa orgulloso ADJ ADJ Gender=Fem|Number=Sing   3   obj _   _
5   de   de   ADP ADP AdpType=Prep   7   case   _   _
6   la   el   DET DET Definite=Def|Gender=Fem|Number=Sing|PronType=Art   7   det _   _
7   actuación actuación NOUN   NOUN   Gender=Fem|Number=Sing   4   nmod   _   _
8   de   de   ADP ADP AdpType=Prep   10   case   _   _
9   su   su   DET DET Number=Sing|Person=3|Poss=Yes|PronType=Prs 10   det _   _
10   hijo   hijo   NOUN   NOUN   Gender=Masc|Number=Sing   7   nmod   _   SpaceAfter=No
11   .   .   PUNCT   PUNCT   PunctType=Peri   3   punct   _   SpaceAfter=No
```

Figura 5. Análisis de oración con aplicación de UDPipe en formato texto

- Tabla:

Id	Form	Lemma	UPosTag	XPosTag	Feats	Head	DepRel	Deps	Misc
# newdoc									
# newpar									
# sent_id = 1									
# text = La madre sonreía orgullosa de la actuación de su hijo.									
1	La	el	DET	DET	Definite=Def|Gender=Fem|Number=Sing|PronType=Art	2	det	_	_
2	madre	madre	NOUN	NOUN	Gender=Fem|Number=Sing	3	nsubj	_	_
3	sonreía	sonreír	VERB	VERB	Mood=Ind|Number=Sing|Person=3|Tense=Imp|VerbForm=Fin	0	root	_	_
4	orgullosa	orgulloso	ADJ	ADJ	Gender=Fem|Number=Sing	3	obj	_	_
5	de	de	ADP	ADP	AdpType=Prep	7	case	_	_
6	la	el	DET	DET	Definite=Def|Gender=Fem|Number=Sing|PronType=Art	7	det	_	_
7	actuación	actuación	NOUN	NOUN	Gender=Fem|Number=Sing	4	nmod	_	_
8	de	de	ADP	ADP	AdpType=Prep	10	case	_	_
9	su	su	DET	DET	Number=Sing|Person=3|Poss=Yes|PronType=Prs	10	det	_	_
10	hijo	hijo	NOUN	NOUN	Gender=Masc|Number=Sing	7	nmod	_	SpaceAfter=No
11	.	.	PUNCT	PUNCT	PunctType=Peri	3	punct	_	SpaceAfter=No

Figura 6. Análisis de oración con aplicación de UDPipe en formato tabla

- Árbol:

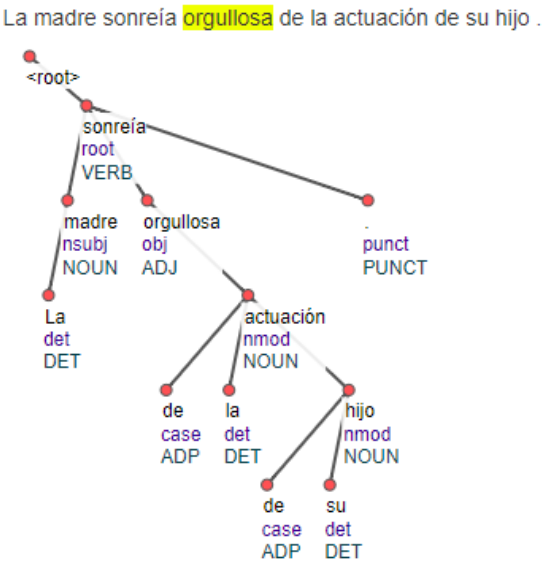


Figura 7. Análisis de oración con aplicación de UDPipe en formato árbol

Además, la aplicación te permite guardar el resultado en un fichero.

2.7.2 LinguaKit

La web Linguakit permite utilizar varias herramientas lingüísticas, entre ellas un analizador sintáctico. Se trata además de un analizador sintáctico de dependencias, al igual que el nuestro.

Fue creado por Cilenis Language Technology, una empresa gallega centrada en transmitir los conocimientos lingüísticos adquiridos sobre el Procesamiento del Lenguaje Natural en la universidad. [7]

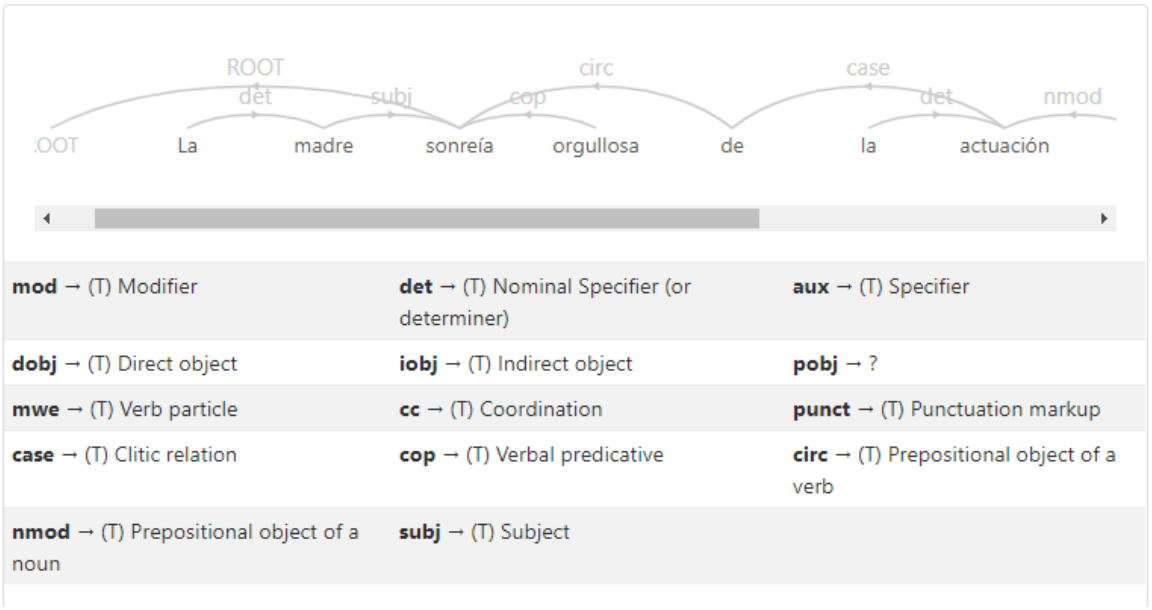
Además de escribir un texto en el recuadro o subir fichero, Linguakit te da también la oportunidad de introducir una URL para analizar su contenido. Otra de sus características es que puede detectar automáticamente el idioma en el que está escrito el texto en cuestión.

Linguakit solo puede analizar frases en cuatro idiomas: español, gallego, portugués e inglés. En este sentido es menos eficiente que el analizador de UDPipe que hemos visto en el apartado anterior.

Linguakit devolverá los análisis que realicemos en dos formatos distintos: grafo y tabla.

Vamos a probar a analizar la misma frase que en los apartados anteriores para ver su funcionamiento.

- Grafo:



mod → (T) Modifier	**det** → (T) Nominal Specifier (or determiner)	**aux** → (T) Specifier
dobj → (T) Direct object	**iobj** → (T) Indirect object	**pobj** → ?
mwe → (T) Verb particle	**cc** → (T) Coordination	**punct** → (T) Punctuation markup
case → (T) Clitic relation	**cop** → (T) Verbal predicative	**circ** → (T) Prepositional object of a verb
nmod → (T) Prepositional object of a noun	**subj** → (T) Subject	

Figura 8. Análisis de oración con Linguakit en formato grafo

- Tabla:

(T) Head position	(T) Head	(T) Head category	(T) Dependent position	(T) Dependent	(T) Dependent category	(T) Syntactic relation
1	madre	nombre	0	el	determinante	Especificador nominal (o determinante)
2	sonreír	verbo	1	madre	nombre	Sujeto
2	sonreír	verbo	3	orgulloso	adjetivo	Atributo verbal
6	actuación	nombre	5	el	determinante	Especificador nominal (o determinante)
2	sonreír	verbo	6	actuación	nombre	Complemento circunstancial
9	hijo	nombre	8	su	determinante	Especificador nominal (o determinante)
6	actuación	nombre	9	hijo	nombre	Complemento nominal

Figura 9. Análisis de oración con Linguakit en formato tabla

Otra de las particularidades de Linguakit es que, si no eres usuario de la aplicación, te permitirá realizar 20 análisis. Para analizar más textos habría que registrarse.

3 Diseño

3.1 Análisis de requisitos

En este apartado vamos a enumerar los requisitos, funcionales y no funcionales, que la aplicación debe satisfacer

3.1.1 Requisitos funcionales

Los requisitos funcionales definen las funciones de un sistema o de sus componentes. Es decir, es el comportamiento del sistema a partir de unas ciertas entradas y las salidas que producirá. Éstos son los requisitos funcionales de la aplicación:

Requisitos funcionales para usuarios no logueados en el sistema:

- **ReqF 01:** Un usuario que no se haya logueado en el sistema, podrá visualizar el historial de oraciones analizadas y el análisis individual de cada una de ellas.

Requisitos funcionales para usuarios logueados en el sistema:

- **ReqF 02:** Un usuario que previamente haya sido añadido por el administrador, podrá acceder al sistema mediante un nombre de usuario y una contraseña.
- **ReqF 03:** Un usuario logueado puede introducir una o varias oraciones en el analizador y elegir con cuál de los dos analizadores quiere analizarlas. También puede elegir los dos analizadores a la vez.
- **ReqF 04:** Un usuario logueado puede, en lugar de introducir oraciones en la caja, subir un fichero con oraciones para analizar. Puede hacerlo arrastrando el fichero a la caja, o mediante el input de debajo (Deberá elegir el analizador también).
- **ReqF 05:** Un usuario logueado podrá, en lugar de introducir oraciones en la caja, introducir una URL. En este caso, el analizador analizará oraciones que haya en el cuerpo de la página web en cuestión.
- **ReqF 06:** Un usuario logueado podrá acceder al historial de frases analizadas.
- **ReqF 07:** Un usuario logueado podrá acceder individualmente a cada frase analizada y ver su análisis en forma de tabla y grafo.
- **ReqF 08:** Un usuario logueado podrá modificar la tabla del análisis de una frase.
- **ReqF 09:** Un usuario logueado podrá salir del sistema cuando desee.

Requisitos funcionales para el administrador del sistema (incluye además los anteriores):

- **ReqF 10:** El administrador del sistema podrá añadir usuarios al sistema introduciendo un nombre de usuario y una contraseña.
- **ReqF 11:** El administrador del sistema podrá limpiar la base de datos del sistema. Esto significa que todas las frases analizadas y todos los usuarios registrados desaparecerán (a excepción del propio administrador).

3.1.2 Requisitos no funcionales

- **ReqNoF 01:** La aplicación web deberá ser compatible con diferentes navegadores y plataformas.
- **ReqNoF 02:** La aplicación deberá ser fácil de usar, sea cual sea el nivel de informática del usuario.
- **ReqNoF 03:** La aplicación deberá ser lo más ligera y rápida posible.
- **ReqNoF 04:** La aplicación debe tener total disponibilidad.

- **ReqNoF 05:** La aplicación debe tener una interfaz de usuario lo más intuitiva posible, de manera que los usuarios puedan utilizarla sin ayuda.

3.2 Casos de uso

En esta subsección vamos a mostrar los diagramas de casos de uso de la aplicación.

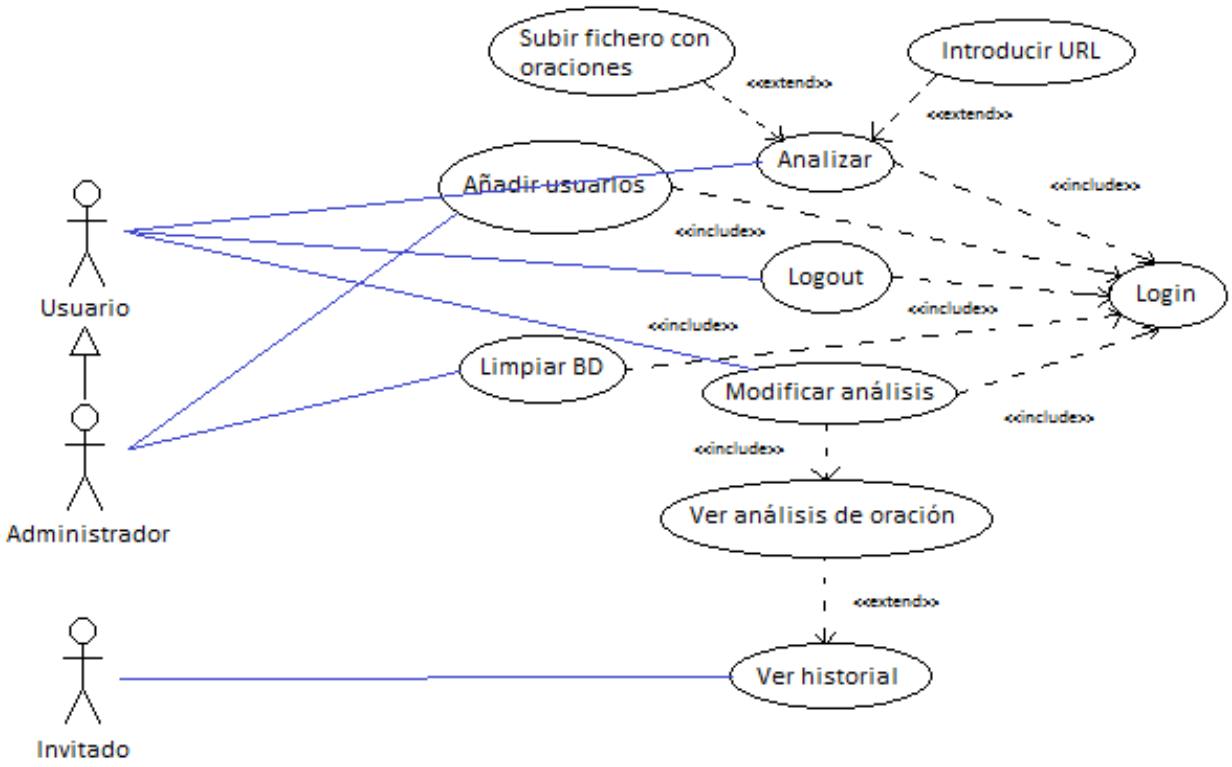


Figura 10. Diagrama de casos de uso

Como vemos, un invitado tan sólo puede ver el historial y los análisis individuales de las oraciones.

Por su parte, un usuario registrado puede, además de acceder al historial y a los análisis de las oraciones, modificar un análisis si cree que éste está mal. También podrá analizar sus propias oraciones, ya sea metiéndolas en la caja de texto, subiendo un fichero que las contenga o extrayéndolas de una URL.

El administrador podrá, además de todo eso, añadir usuarios a la base de datos y limpiar ésta, de manera que se borren todas las oraciones almacenadas y también los usuarios a excepción del propio administrador.

3.3 Tecnologías usadas

En esta subsección se explican las tecnologías y lenguajes de programación usados para implementar la aplicación.

3.3.1 JSP

La aplicación se ha implementado usando JSP (Java Servlet Pages).

Las JSP tienen una gran ventaja, permite meter pequeños fragmentos de lenguaje Java en medio de un fichero HTML. Esto hace que podamos separar las partes encargadas de generar las páginas web en código HTML, de las partes encargadas de la funcionalidad de la aplicación, que estarán programadas en código Java. Estas partes son llamadas Servlets.

Esto hace que el código sea más limpio y eficiente, ya que de otro modo, tendríamos que meter el código HTML mezclado con el código Java.

Otras ventajas de las JSP son las siguientes:

- Fáciles de entender y de mantener
- No se necesita recompilar ni redesplegar en caso de cambiar algo.
- Tienen menos código que el servlet.

Al usar JSPs, hemos usado también los llamados Java Beans. Los Java Beans son clases Java que tienen las siguientes propiedades:

- Implementan la interfaz Serializable.
- Tienen un cierto número de atributos que, pueden ser de cualquier clase Java y que han de ser privados.
- El constructor debe ser vacío.
- Tiene getters y setters para sus atributos.

En la siguiente imagen podemos ver el funcionamiento de las JSPs, los Servlets y los Beans:

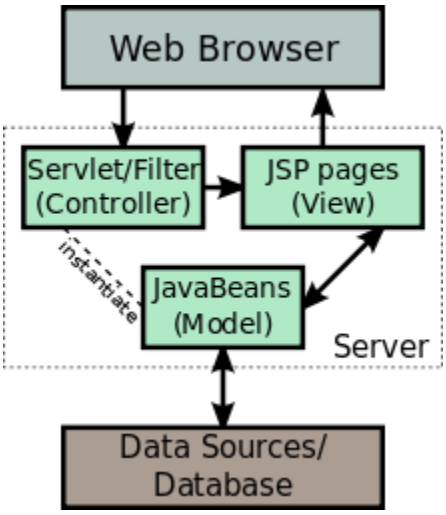


Figura 11. Funcionamiento de los JSP

Como podemos ver en la imagen, se sigue el patrón Modelo-Vista-Controlador.
El patrón nace con el objetivo de reducir el esfuerzo de programación a partir de estandarizar el diseño de las aplicaciones [3].

Este sistema divide las aplicaciones en tres partes: Modelo, Vista y Controlador. Cada parte se puede implementar por separado.

En la anterior imagen queda claro cuál es cada parte. Los Java Beans son el Modelo, las páginas JSP son la vista (recordemos que son las que contienen el código HTML) y los Servlets son el Controlador (como hemos comentado antes, en ellos se encuentra la funcionalidad de la aplicación).

- **Modelo**: se trata de la parte que contiene los datos de la aplicación. Como hemos comentado antes, tiene funciones para acceder a dichos datos.
- **Vista**: se trata de la parte que contiene la interfaz de usuario. Como ya hemos dicho antes, en nuestra aplicación son las páginas JSP, que contienen el código HTML.
- **Controlador**: se trata de la parte de la aplicación que contiene la funcionalidad para responder a las acciones que se le solicitan.

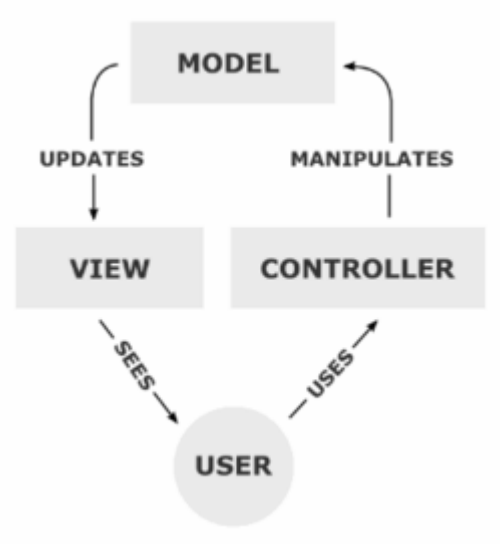


Figura 12. Esquema del patrón Modelo-Vista-Controlador

3.3.2 Apache Derby DB

El sistema utiliza una base de datos para gestionar las frases que se analizan y también los usuarios que tienen acceso al sistema.

Para esta aplicación hemos utilizado **Apache Derby DB**. Se trata de un gestor de bases de datos implementado enteramente en lenguaje Java y que proporciona un driver JDBC. Se trata de un gestor ideal para usarlo en aplicaciones Java como la nuestra, ya que no tiene bindings para ningún otro lenguaje.

3.3.3 Graphviz

Cuando accedemos al sistema y analizamos una oración, además del análisis en forma de tabla, la aplicación nos mostrará un análisis en forma de grafo.

Para poder crear los grafos, hemos optado por utilizar la herramienta **Graphviz**. Se trata de un software de código abierto que, mediante un comando sencillo, crea un grafo y da la posibilidad de importarlo en varios formatos, por ejemplo en .jpg o .png.

Para ello, hay que pasarle un fichero de texto con un formato concreto, mediante el cual creará el grafo. En la siguiente imagen se puede ver un ejemplo de fichero:

```
digraph G
{
    "Esto";
    "es";
    "un";
    "ejemplo";
    "para";
    "el";
    "proyecto";
    ROOT;

    "Esto" -> "ejemplo" [label="nsubj"];
    "es" -> "ejemplo" [label="cop"];
    "un" -> "ejemplo" [label="det"];
    "ejemplo" -> ROOT [label="root"];
    "para" -> "proyecto" [label="case"];
    "el" -> "proyecto" [label="det"];
    "proyecto" -> "ejemplo" [label="nmod"];
}
```

Figura 13 . Ejemplo de fichero de entrada para Graphviz

Como podemos ver, la estructura del fichero es la siguiente:

- Al principio hay que poner "digraph G" y abrir llaves, donde irá todo el contenido del grafo.
- Dentro de las llaves hay que poner, una detrás de otra y separadas por ";", las etiquetas que irán en cada nodo del grafo. En este caso, se trata de las palabras de la frase.
- Por último hay poner las flechas desde el nodo 1 al nodo 2 y poner entre corchetes la etiqueta que acompañará a la arista correspondiente.

Para sacar el grafo a partir de dicho fichero, habría que ejecutar un comando como el siguiente:

```
C:\Users\TRJJUR\Desktop\release\bin>dot.exe -Tjpg ..\..\TFGv3\web\graph1.txt -o grafo.jpg
```

Figura 14. Comando ejecución Graphviz

Si lo hemos hecho bien, se habrá creado un fichero "grafo.jpg" que tendrá el siguiente aspecto:

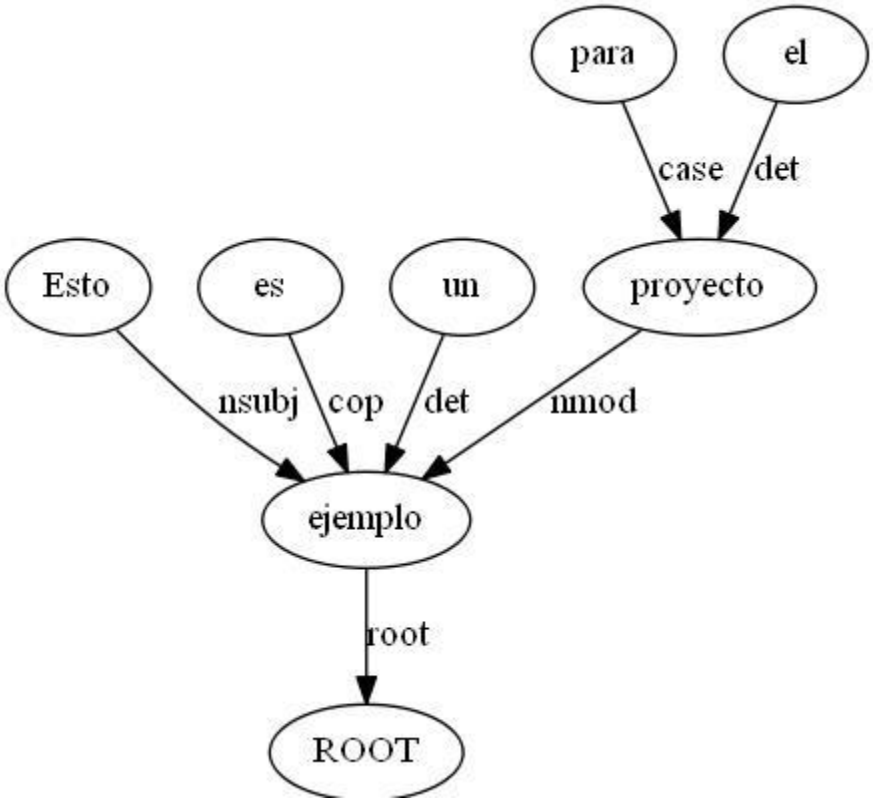


Figura 15. Ejemplo de grafo creado por Graphviz

4 Desarrollo

En esta sección se va a explicar en detalle la implementación del código creado para el funcionamiento de la aplicación y el diseño de la base de datos que hemos creado. Como ya se ha explicado en el apartado *Diseño*, la parte correspondiente al modelo y al controlador se ha implementado en lenguaje Java, mientras que la parte correspondiente a la vista se ha implementado usando páginas JSP, que contienen código HTML, JavaScript y Java.

4.1 Diseño de la base de datos

La base de datos que hemos creado está compuesta por cuatro tablas que son las siguientes:

- **USUARIOS**: esta tabla contiene la información de los usuarios registrados en el sistema. Esta tabla está compuesta por tres campos:
    - *USERNAME*: cadena de caracteres que indica el nombre de usuario. Este campo es la clave primaria de la tabla.
    - *PASSWORD*: cadena de caracteres que indica la contraseña con la que el usuario podrá acceder al sistema.
    - *ROLE*: cadena de caracteres que indica el rol del usuario en la aplicación. Puede ser de dos tipos: **Admin** y **User**. El administrador del sistema es el único que tiene el rol **Admin**, el resto de usuarios a los que registre en la aplicación se añadirán con rol **User**.

- **FRASE_ORIGINAL:** esta tabla contiene las frases originales, tal y como se metieron a la aplicación para analizarlas. Esta tabla está compuesta por cinco campos:
    - *FRASE_ORIG_ID:* número entero que sirve como identificador único para cada una de las oraciones. Este campo es la clave primaria de la tabla.
    - *FRASE:* cadena de caracteres que contiene la frase que el usuario a metido en el sistema para analizarla.
    - *MODIFICADA:* booleano que indica si la frase ha sido modificada o no.
    - *SPACY:* booleano que indica si la frase ha sido analizada usando el analizador SpaCy o no.
    - *UDPIPE:* booleano que indica si la frase ha sido analizada usando el analizador UDPipe o no.

- **TABLA_ORIGINAL:** esta tabla contiene cada una de las palabras de las frases que se analizan en el sistema. Guarda todos los campos del formato CoNLL-U para cada palabra. Esta tabla está compuesta por los siguientes doce campos:
    - *TABLA_ORIG_ID:* numero entero que sirve como identificador único para cada una de las palabras que hay en el sistema. Este campo es la clave primaria de la tabla.
    - *ID:* número entero que conforma uno de los campos del formato CoNLL-U. Indica el índice de la palabra dentro de la frase.
    - *FORM:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene la forma de la palabra.
    - *LEMMA:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene el lema de la forma de la palabra.

- *UPOS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene la etiqueta part-of-speech universal.
- *XPOS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene la etiqueta part-of-speech específica para cada idioma.
- *FEATS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene una lista de características morfológicas del inventario de características universal o de una extensión específica del idioma.
- *HEAD:* número entero que conforma uno de los campos del formato CoNLL-U. Indica el índice de la palabra con la que se relaciona la palabra actual.
- *DEPREL:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Indica qué relación existe entre las dos palabras (la actual y la palabra apuntada por el campo HEAD).
- *DEPS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene una lista de pares HEAD-DEPREL.
- *MISC*: cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene cualquier otra anotación sobre la palabra.
- *FRASE_ORIG_ID*: número entero que actúa como clave externa de la tabla. Hace referencia a la clave primaria de la tabla **FRASE_ORIGINAL**.

- **TABLA_MODIFICADA:** esta tabla contiene cada una de las palabras de las frases del sistema, una vez han sido modificadas. Es bastante similar a la tabla **TABLA_ORIGINAL**, ya que también contiene todos los campos del formato CoNLL-U. Esta tabla tiene un campo que la anterior, un total de trece:
  - *TABLA_MOD_ID:* numero entero que sirve como identificador único para cada una de las palabras que hay en la tabla. Este campo es la clave primaria de la tabla.
  - *ID:* número entero que conforma uno de los campos del formato CoNLL-U. Indica el índice de la palabra dentro de la frase.
  - *FORM:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene la forma de la palabra.
  - *LEMMA:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene el lema de la forma de la palabra.
  - *UPOS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene la etiqueta part-of-speech universal.
  - *XPOS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene la etiqueta part-of-speech específica para cada idioma.
  - *FEATS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene una lista de características morfológicas del inventario de características universal o de una extensión específica del idioma.
  - *HEAD:* número entero que conforma uno de los campos del formato CoNLL-U. Indica el índice de la palabra con la que se relaciona la palabra actual.
  - *DEPREL:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Indica qué relación existe entre las dos palabras (la actual y la palabra apuntada por el campo HEAD).
  - *DEPS:* cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene una lista de pares HEAD-DEPREL.
  - *MISC*: cadena de caracteres que conforma otro de los campos del formato CoNLL-U. Contiene cualquier otra anotación sobre la palabra.
  - *FRASE_ORIG_ID*: número entero que actúa como clave externa de la tabla. Hace referencia a la clave primaria de la tabla **FRASE_ORIGINAL**.

- *TABLA_ORIG_ID*: número entero que actúa como clave externa de la tabla. Hace referencia a la clave primaria de la tabla **TABLA_ORIGINAL**.

A continuación se muestra el modelo relacional de la base de datos:

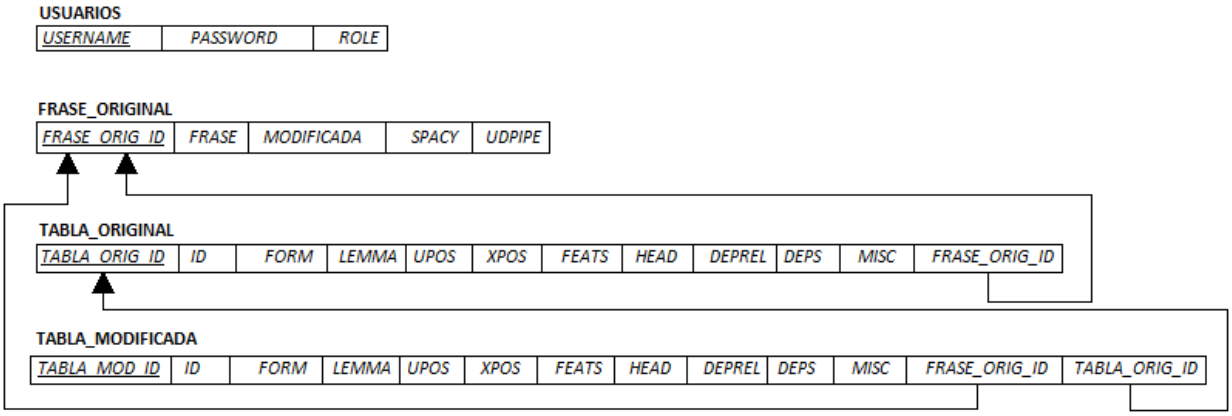


Figura 16. Modelo Relacional BD

4.2 Páginas JSP

Vamos a profundizar en el código de las páginas JSP que contiene la aplicación. Todas las páginas JSP están creadas utilizando *templates* de la herramienta *Bootstrap*.

4.2.1 Login.jsp

Esta página será la que nos dé la bienvenida, una vez iniciada la aplicación. Para que esto sea así, lo hemos configurado en el fichero *web.xml* del proyecto:

```
<welcome-file-list>
    <welcome-file>JSP/Login.jsp</welcome-file>
</welcome-file-list>
```

Figura 17. Welcome-file web.xml

Esta página contiene dos formularios, correspondientes cada uno a una de las pestañas superiores:

- **Formulario "Login"**: este formulario sirve para acceder a la aplicación introduciendo el nombre de usuario y la contraseña con las que has sido registrado por el administrador. En caso de tratarse del propio administrador podrá acceder introduciendo *admin* como usuario y como contraseña.

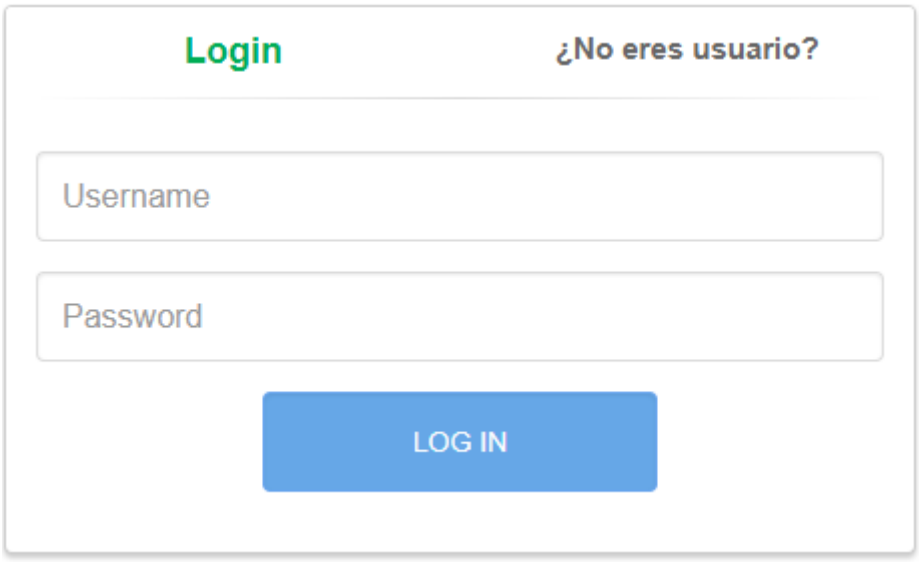


Figura 18. Login.jsp - Formulario "Login"

- **Formulario "¿No eres usuario?":** si pinchamos en la pestaña "¿No eres usuario?" el formulario de "Login" desaparecerá para mostrar otro formulario. Éste contiene solamente un botón que nos permitirá ver el historial de frases analizadas en caso de que no estemos registrados en la aplicación.

Figura 19. Login.jsp - Formulario "¿No eres usuario"

4.2.2 NonUser.jsp

Esta página aparecerá si accedemos al sistema sin autenticarnos, es decir, si pulsamos el botón *Ver Historial (Figura 10).* En ella se muestran todas las frases que hay actualmente analizadas en la base de datos del sistema.

Figura 20. NonUser.jsp

Como podemos ver, en la página aparece el menú lateral con la opción de autenticarse en el sistema. En la sección principal de la página estarán las frases analizadas por el sistema. Se puede observar que en cada frase hay 3 *checks* que indican:

- Si la frase ha sido modificado por algún usuario.
- Si la frase ha sido analizada usando el analizador SpaCy.
- Si la frase ha sido analizada usando el analizador UDPipe.

Por último, cada frase tiene también un botón *Ver*, que nos permitirá ver el análisis que ha hecho el sistema de dicha frase.

4.2.3 PhraseViewGuest.jsp

Si en la página *NonUser.jsp* pulsamos sobre el botón *Ver* de una frase, llegaremos a esta página. La Figura 12 muestra, a modo de ejemplo, el resultado de analizar la frase *Esto es una prueba para probar el analizador.*

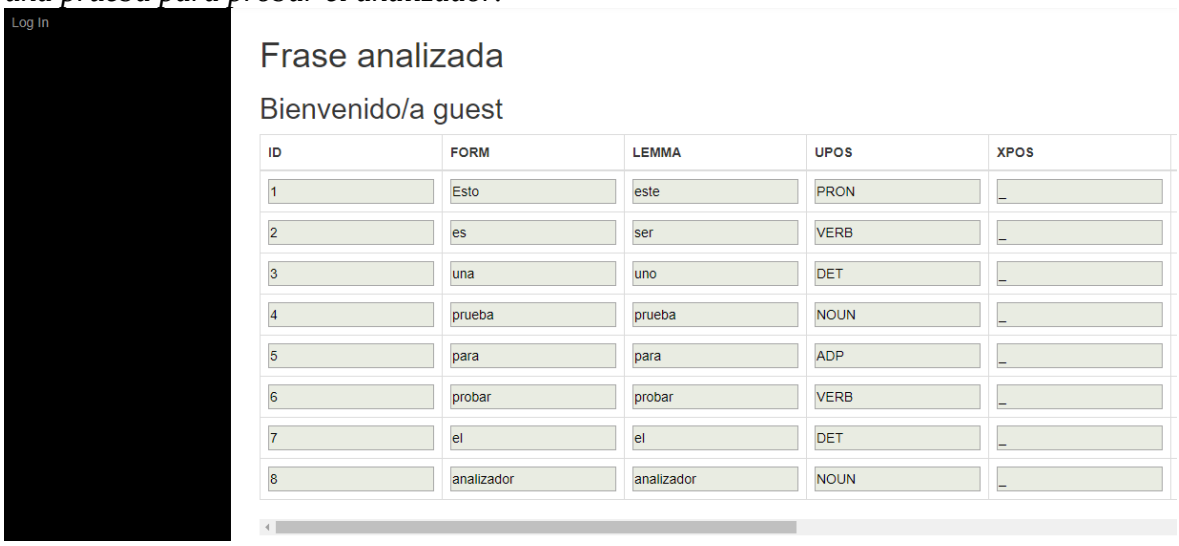


Figura 21. PhraseViewGuest.jsp - Tabla

En la imagen podemos ver la tabla con el análisis de la frase. La tabla contiene los campos del formato CoNLL-U. En la imagen no podemos verlos todos, ya que habría que correr la *scrollBar* hacia la derecha.

Si bajamos en la página, podremos ver el grafo obtenido del análisis:

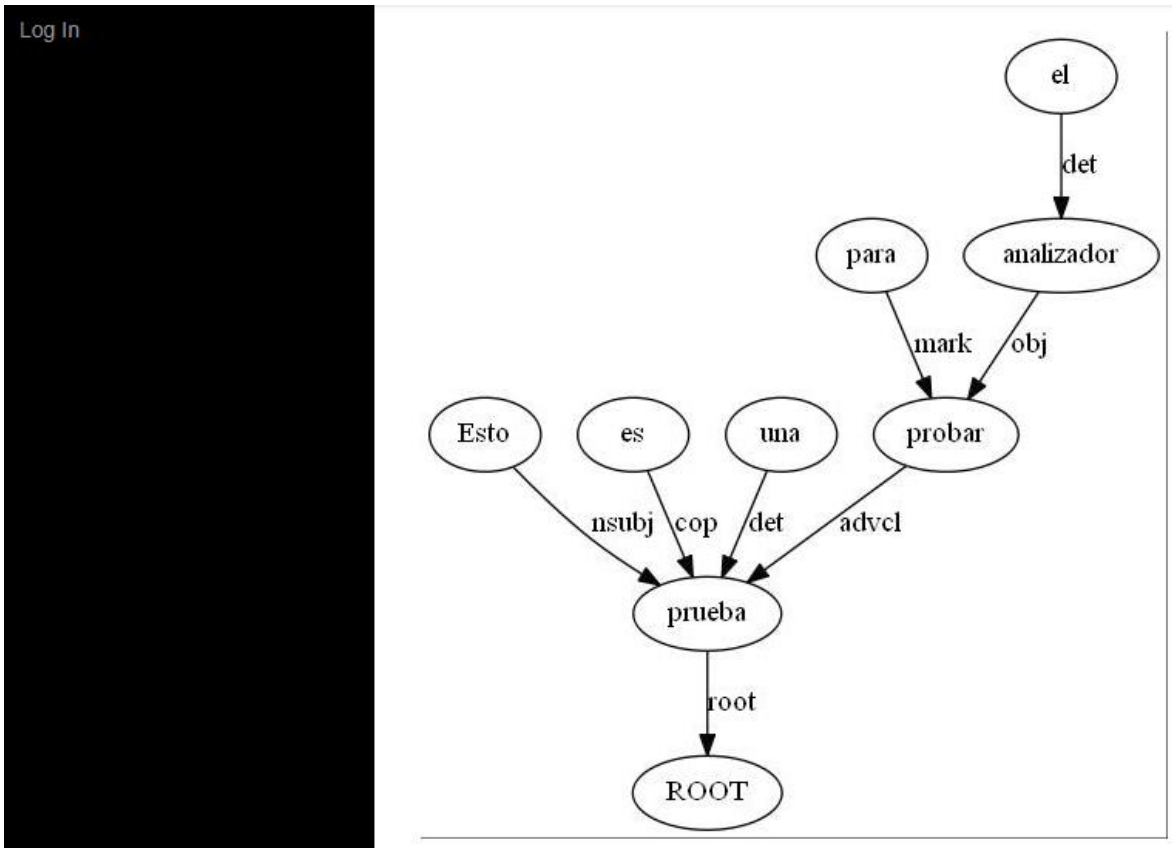


Figura 22. PhraseViewGuest.jsp - Grafo

Debajo del grafo encontraremos un botón para volver al historial.

4.2.4 Admin.jsp

Si nos autenticamos en el sistema, como administrador o como usuario, llegaremos a esta página, en la que encontraremos varios elementos:

- **Menú lateral:** podemos observar que en el menú lateral ahora aparecen varias opciones:
  - *Parser:* hacer clic en esta opción nos llevaría a la misma página en la que nos encontramos.
  - *Historial:* esto nos llevaría al historial de frases analizadas, al igual que si entrábamos sin autenticarnos.
  - *Log Out:* si pinchamos, saldremos del sistema.
  - *Añadir Usuario:* esta pestaña sólo le aparece al administrador. Con ella podremos acceder a la pantalla para añadir a un nuevo usuario al sistema.
  - *Limpiar BD:* esta pestaña también le aparece sólo al administrador. Con ella podremos eliminar todas las frases analizadas y usuarios de la base de datos.
- **Caja de texto:** La mayor parte de la página la ocupa un *input* para introducir texto. En él, podremos escribir las frases que queramos analizar. Como ya hemos comentado anteriormente, también podremos introducir una URL o bien arrastrar dentro un fichero de texto. Si nos fijamos en la esquina inferior derecha de la caja, vemos que podemos redimensionar el alto de la caja.
- **Elección de analizador:** debajo de la caja del texto, habrá dos *checkboxes* con los que indicaremos si queremos analizar la frase con el analizador SpaCy, con el UDPipe o con los dos. Tras indicar el analizador que deseemos, haremos clic en **Parsear** y obtendremos el análisis de la frase.

- **Subida de fichero:** además de arrastrar un fichero de texto a la caja, también podemos subir un fichero por aquí. Para ello hacemos clic en "Seleccionar archivo" y elegiremos el fichero de texto que deseemos. Después elegiremos el analizador que queramos y hacemos clic en **Subir**. Así obtendremos el análisis del texto del fichero.

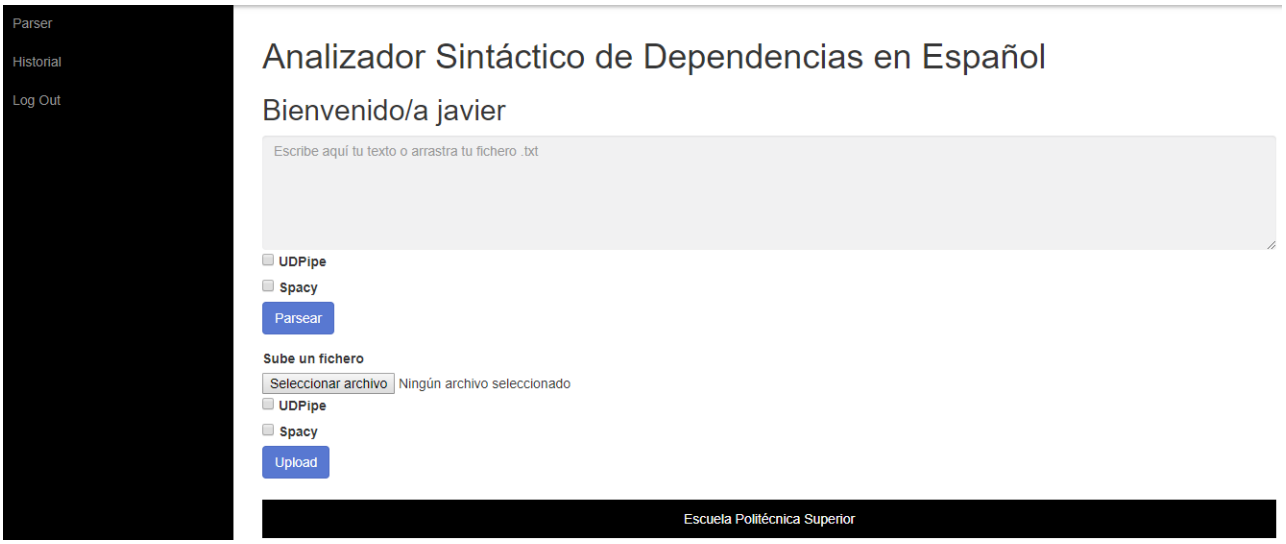


Figura 23. Admin.jsp

4.2.5 Parsed.jsp

Si analizamos una o varias frases, llegaremos a esta página. El aspecto de esta página es muy similar al de *PhraseViewGuest.jsp* aunque tiene algunas diferencias. La principal es que, en la tabla, podemos modificar los resultados obtenidos. Todos los campos del formato CoNLL-U son editables y, a la derecha del todo de cada palabra, tendremos un botón para guardar los campos. Además, en este caso puede aparecer más de una frase dependiendo de cuántas metamos en el analizador.

4.2.6 ModifiedPage.jsp

Si modificamos alguna palabra de alguna frase, llegaremos a esta página. Se trata de una página muy simple que solamente muestra un mensaje de confirmación de que se ha modificado correctamente la palabra. Por supuesto, cuenta también con el menú lateral y con un botón *Volver* que nos llevará a la página del historial.

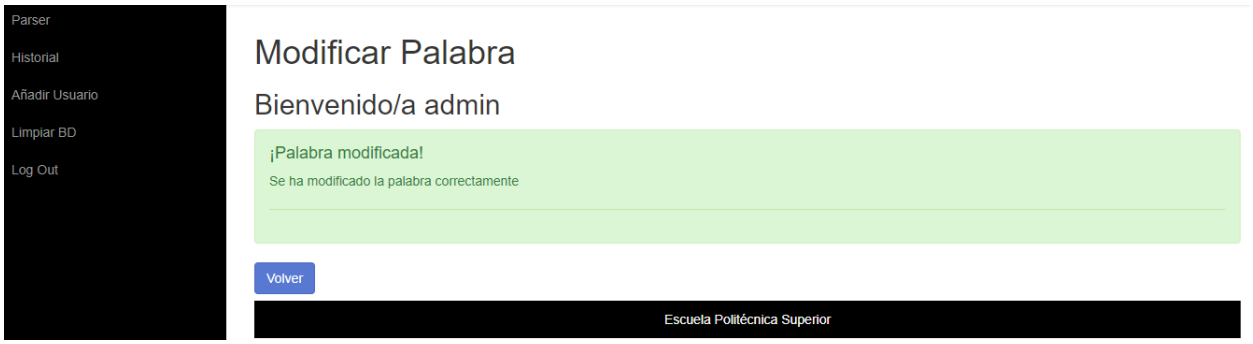


Figura 24. ModifiedPage.jsp

4.2.7 Modify.jsp

Esta página contiene el historial de frases analizadas por el sistema. Es una página prácticamente igual que *NonUser.jsp*, a la que llegábamos si entrábamos sin autenticarnos en el sistema. Sólo cambiaría el menú lateral, ya que ahora mostraría las opciones propias de un usuario o administrador.

Figura 25. Modify.jsp

4.2.8 PhraseView.jsp

Esta página también se asemeja mucho a *PhraseViewGuest.jsp*. Esta vez, al igual que en *Parsed.jsp*, los campos del formato CoNLL-U son editables, ya que sólo podrán acceder a esta página los usuarios registrados y el administrador, que son los que pueden editar frases.

4.2.9 AddUser.jsp

Esta página le aparecerá al administrador del sistema si hace clic en el apartado *Añadir Usuario* del menú lateral. Contiene un pequeño formulario que se utilizará para dar de alta a nuevos usuarios en la aplicación. El formulario consta de tres campos:

- El nombre de usuario que tendrá el nuevo usuario de la aplicación. Este campo no puede ser igual en dos usuarios
- La contraseña que tendrá el usuario para poder acceder a la aplicación.
- La confirmación de la contraseña del usuario. Si la contraseña y la confirmación no coinciden, no podremos añadir al usuario.

Por último habrá un botón *Registrar* para mandar la información. Este botón estará deshabilitado si, como hemos comentado, las dos contraseñas no coinciden.

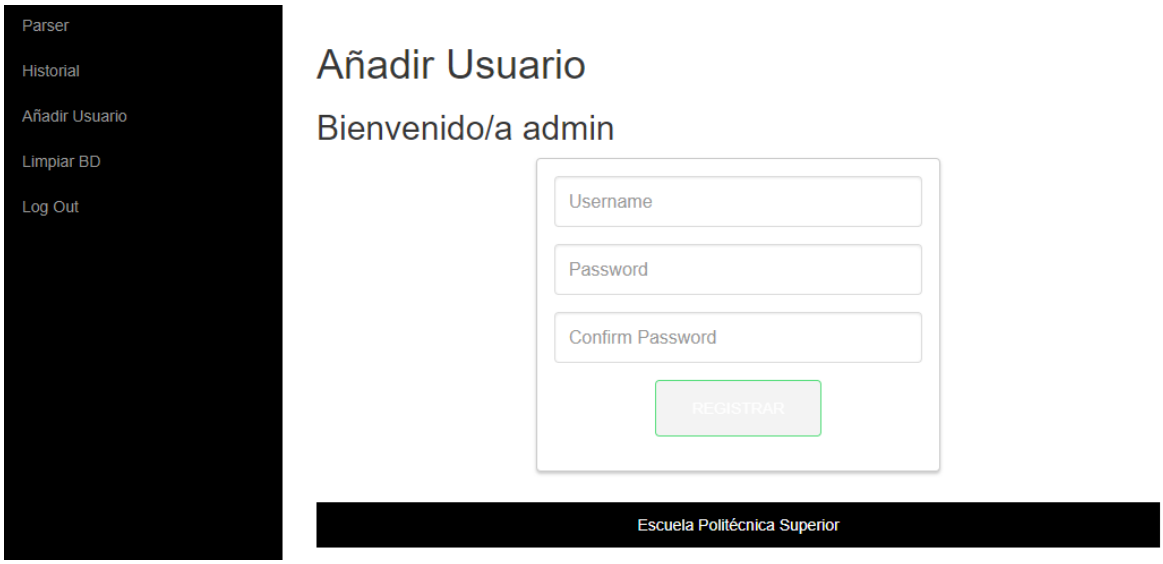


Figura 26. AddUser.jsp

4.2.10 Added.jsp

Esta página, al igual que *ModifiedPage.jsp* muestra solamente un mensaje de confirmación. Aparecerá cuando añadamos un nuevo usuario al sistema. Si hacemos clic en el botón *Volver*, seremos redireccionados a la página principal, es decir, *Admin.jsp*.

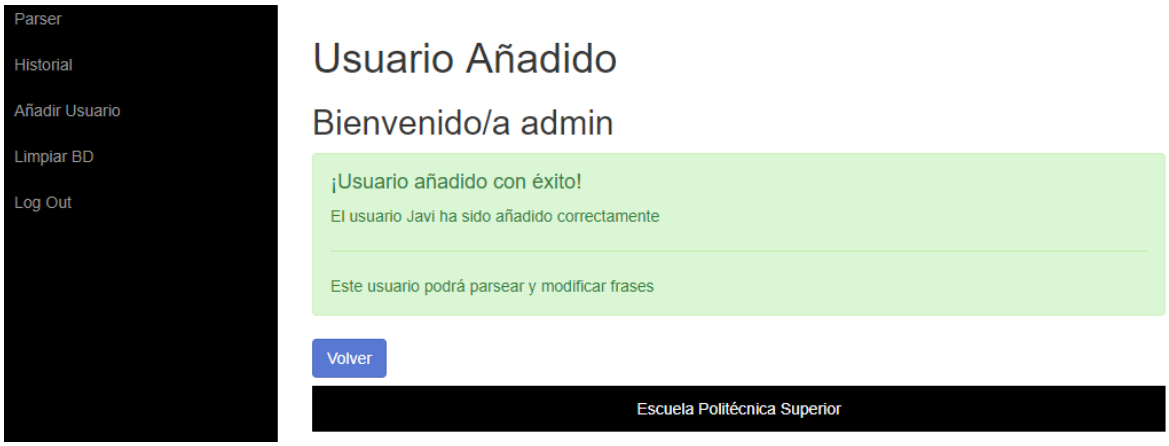


Figura 27. Added.jsp

4.2.11 CleanBD.jsp

Esta página también muestra un mensaje de confirmación. En este caso, llegaremos a esta página cuando hacemos clic en la pestaña *Limpiar BD*. Esto provocará que se borren todas las frases y usuarios de la aplicación, a excepción del usuario *admin*.

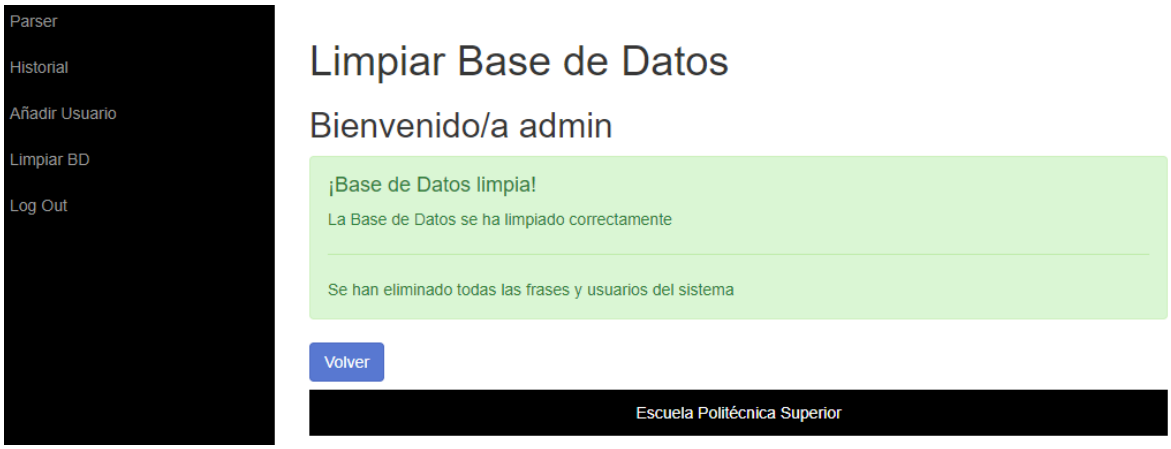


Figura 28. CleanBD.jsp

4.2.12 NoCheckbox.jsp

Esta página contiene un mensaje de error. Llegaremos a ella cuando intentemos analizar oraciones sin haber seleccionado ningún analizador para ello. Al hacer clic en el botón *Volver*, se nos redireccionará a la página principal, *Admin.jsp*.

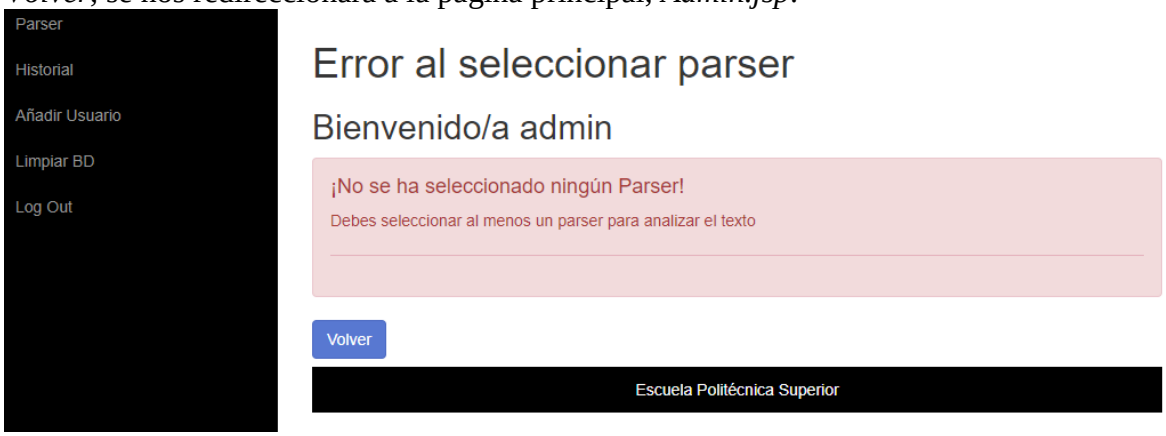


Figura 29. NoCheckbox.jsp

4.2.13 NoFile.jsp

Esta página es muy parecida a *NoCheckbox.jsp*. También muestra un mensaje de error, aunque en este caso aparecerá cuando queramos subir un fichero sin haber seleccionado ninguno antes. El botón *Volver* nos volverá a llevar a la página principal.

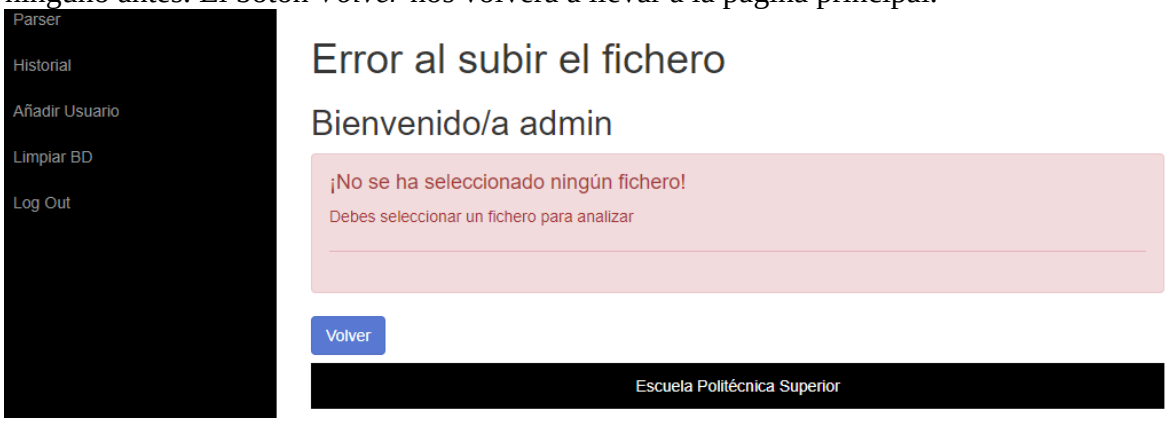


Figura 30. NoFile.jsp

4.3 Java Beans

Como ya hemos comentado en el apartado 3.3.1, hemos utilizado Java Beans en nuestro proyecto. Uno por cada tabla de la base de datos, además de uno que utilizamos como clase abstracta. Los hemos situado en el paquete *spanish.parser.beans*. Son los siguientes:

4.3.1 FraseOriginalBean.java

Este Java Bean corresponde a la tabla FRASE_ORIGINAL y contiene los mismos campos:

- Frase_orig_id: entero que contiene el identificador de la frase que actúa como clave primaria en la tabla.
- Frase: cadena de caracteres que contiene la frase original.
- Modificada: booleano que indica si la frase se ha modificado o no.
- Spacy: booleano que indica si la frase ha sido analizada con el analizador Spacy o no.
- Udpipe: booleano que indica si la frase ha sido analizada con el analizador UDPipe o no.

4.3.2 PalabraBean.java

En el paquete *spanish.parser.beans*, además de los Java Beans correspondiente a las cuatro tablas de la base de datos, hemos creado esta clase abstracta que servirá para que las clases *PalabraOriginalBean.java* y *PalabraModificadaBean.java*, correspondientes a dos de las tablas, hereden de ella, ya que son muy parecidas. En esta clase abstracta meteremos todos los atributos que estas dos clases tienen en común, es decir, todos los campos del formato CoNLL-U y la clave externa *frase_orig_id.* Contiene los siguientes atributos:

- Id: entero que contiene el campo ID del formato CoNLL-U para la palabra tratada.
- Head: entero que contiene el campo HEAD del formato CoNLL-U para la palabra tratada.
- Frase_orig_id: entero que contiene el identificador de la frase en la que está contenida la palabra. Correspondiente a una clave externa en la base de datos.
- Form: cadena de caracteres que contiene el campo FORM del formato CoNLL-U para la palabra tratada.
- Lemma: cadena de caracteres que contiene el campo LEMMA del formato CoNLL-U para la palabra tratada.
- Upos: cadena de caracteres que contiene el campo UPOS del formato CoNLL-U para la palabra tratada.
- Xpos: cadena de caracteres que contiene el campo XPOS del formato CoNLL-U para la palabra tratada.
- Feats: cadena de caracteres que contiene el campo FEATS del formato CoNLL-U para la palabra tratada.
- Deprel: cadena de caracteres que contiene el campo DEPREL del formato CoNLL-U para la palabra tratada.
- Deps: cadena de caracteres que contiene el campo DEPS del formato CoNLL-U para la palabra tratada.
- Misc: cadena de caracteres que contiene el campo MISC del formato CoNLL-U para la palabra tratada.

En esta misma clase se han implementado ya los getters y los setters de estos atributos.

4.3.3 PalabraOriginalBean.java

Este Java Bean hereda de *PalabraBean.java* y se encuentra también en el paquete *spanish.parser.beans*. Corresponde a la tabla PALABRA_ORIGINAL de la base de datos. Además de todos los atributos de dicha clase, contiene un atributo *tabla_orig_id*:

- Tabla_orig_id: entero que sirve de identificador de la palabra original. Corresponde a la clave primaria en la tabla de la base de datos.

Se han implementado el getter y el setter del anterior atributo.

4.3.4 PalabraModificadaBean.java

Este Java Bean también hereda de *PalabraBean.java* y se encuentra también en el paquete *spanish.parser.beans*. Corresponde a la tabla PALABRA_MODIFICADA de la base de datos. Además de todos los atributos de dicha clase, contiene un atributo *tabla_mod_id*:

- Tabla_mod_id: entero que sirve de identificador de la palabra modificada. Corresponde a la clave primaria en la tabla de la base de datos.
- Tabla_orig_id: entero que sirve de identificador de la palabra original que ha sido modificada. Corresponde a una clave externa en la tabla de la base de datos.

Se han implementado los getters y los setters para estos atributos.

4.3.5 UserBean.java

Este Java Bean corresponde a la tabla USUARIOS de la base de datos. En el apartado 3.3.2 hemos comentado que dicha tabla contiene tres campos: *username*, *password* y *role*. En este caso, en el Java Bean solamente hemos declarado dos atributos: *username* y *password*. Esto es debido a que no nos hacía falta implementar el atributo *role*.

- UserName: cadena de caracteres que contiene el nombre de un usuario.
- Password: cadena de caracteres que contiene la contraseña del usuario.

Este Bean se utiliza para autenticarse en la aplicación.

4.4 Util

Hemos creado un paquete llamado *spanish.parser.util* en el que hemos implementado varias clases que nos ayudarán a la hora de implementar los servlets. Se trata de clases que servirán para tareas tales como el manejo de la base de datos, la generación de grafos o el funcionamiento de los analizadores. Vamos a profundizar un poco en cada una de ellas:

4.4.1 MyProperties.java

Esta clase se utiliza para acceder a las propiedades del proyecto, almacenadas en un fichero *parser.properties*. Esta clase lee las propiedades y las guarda en un *HashMap*. Podremos obtener la propiedades que queramos mediante el método *getProperty (String key)*, al que le pasamos la clave y nos devolverá el valor de dicha propiedad.

4.4.2 ParserType.java

Esta clase no es tal, sino que es una enumeración que contiene los tipos de analizadores que hay que en el sistema. Actualmente son dos: UDPipe y Spacy.

```
public enum ParserType {
    UDPIPE, SPACY
}
```

Figura 31. ParserType.java

4.4.3 Parser.java

Esta clase es abstracta, al igual que ocurría con *PalabraBean.java*. Cada analizador heredará de esta clase, que contiene como atributo, el tipo de analizador que es (utilizando la enumeración *ParserType*). Además, esta clase contiene dos métodos abstractos, que cada uno de los analizadores que contenga el sistema tendrán que implementar. Se trata de los siguientes métodos:

- *getConlluParse (String text, ServletContext context)*: esta función obtendrá el análisis en formato CoNLL-U del texto *text* que le pasemos como argumento y lo devolverá en forma de cadena de caracteres.
- g*etWordsFromPhrase (String text)*: esta función procesará el texto que le devuelve la anterior, y devuelve una lista de *PalabraBeans* con el análisis de la oración en cuestión.

4.4.4 SpacyParser.java

Esta clase es una de las que heredarán de la anterior clase abstracta. Esta clase implementa las funciones para el analizador Spacy. Solamente contiene dos métodos que no son otros que los dos métodos abstractos que declaramos en *Parser.java*, algo obligatorio por la herencia.

- En el método *getConlluParse*, se ejecuta el análisis en Python de la oración mediante línea de comandos, usando la clase *Runtime*. Después recogeremos en una cadena de caracteres el resultado del análisis.
- En el método *getWordsFromPhrase*, tokenizaremos la cadena de caracteres que obtenemos en el anterior método (clase *StringTokenizer*) e iremos creando *PalabraBeans* que al final devolveremos en una lista.

```java
@Override
public List<PalabraBean> getWordsFromPhrase(String text) {
    List<PalabraBean> lista = new ArrayList<>();
    String delim_linea = "\n";
    String delim_campo = "\t";

    StringTokenizer st1 = new StringTokenizer(text, delim_linea);
    while (st1.hasMoreTokens()) {
        StringTokenizer st2 = new StringTokenizer(st1.nextToken().trim(), delim_campo);
        PalabraBean palabra = new PalabraOriginalBean();
        palabra.setId(Integer.parseInt(st2.nextToken()));
        palabra.setForm(st2.nextToken());
        palabra.setLemma(st2.nextToken());
        palabra.setUpos(st2.nextToken());
        palabra.setXpos(st2.nextToken());
        palabra.setFeats(st2.nextToken());
        palabra.setHead(Integer.parseInt(st2.nextToken()));
        palabra.setDeprel(st2.nextToken());
        palabra.setDeps(st2.nextToken());
        palabra.setMisc(st2.nextToken());
        lista.add(palabra);
    }
    return lista;
}
```

Figura 32. getWordsFromPhrase - SpacyParser.java

4.4.5 UDPipeParser.java

Esta clase es similar a la anterior, pero esta vez implementa el analizador UDPipe. Mientras que en el analizador Spacy teníamos que ejecutar por consola un comando Python, en el analizador UDPipe tendremos una librería *udpipe.jar* que contendrá las clases y métodos necesarios en Java para analizar las oraciones. Contiene los dos métodos abstractos de la clase *Parser.java*:

- En el método *getConlluParse* tendremos que utilizar las clases de la librería *udpipe.jar* para poder analizar la frase. Primero tendremos que cargar el modelo usando el método *load* de la clase *Model*. Después analizaremos la frase mediante la clase *Pipeline* usando el método *process*, que nos devolverá una cadena con el análisis. Dicha cadena será la que devolvamos.

```java
@Override
public String getConlluParse(String text, ServletContext context) {
    udpipe_java.setLibraryPath(System.getenv("GLASSFISH_HOME") + MyProperties.getProperty("dllPath"));
    String modelFile = MyProperties.getProperty("udModelPath");

    String input = "horizontal";
    String output = "conllu";
    Model model = Model.load(context.getRealPath("/WEB-INF") + modelFile);
    if (model == null) {
        System.out.println("Cannot load model from file '" + modelFile + "'");
        return null;
    }

    System.out.println("done\n");

    Pipeline pipeline = new Pipeline(model, input, Pipeline.getDEFAULT(), Pipeline.getDEFAULT(),
            output);
    ProcessingError error = new ProcessingError();

    String processed = pipeline.process(text, error);

    if (error.occurred()) {
        System.out.println("Cannot read input CoNLL-U: " + error.getMessage());
        return null;
    }

    model = null;
    System.out.println(processed);
    return processed;
}
```

Figura 33. getConlluParse - UDPipeParser.java

- En el método *getWordsFromPhrase* haremos lo mismo que en *SpacyParser.java*, es decir, tokenizar la cadena que nos devuelve la función anterior y crea una lista de *PalabraBeans* que guarden los campos del formato CoNLL-U de cada palabra. Al final devolveremos dicha lista.

```java
@Override
public List<PalabraBean> getWordsFromPhrase(String text) {
    List<PalabraBean> lista = new ArrayList<>();
    String delete = "# newdoc\n" + "# newpar\n" + "# sent_id = ";
    String delim_linea = "\n";
    String delim_campo = "\t";

    StringTokenizer st1 = new StringTokenizer(text.replaceAll(delete, ""), delim_linea);
    st1.nextToken();
    while (st1.hasMoreTokens()) {
        StringTokenizer st2 = new StringTokenizer(st1.nextToken().trim(), delim_campo);
        PalabraBean palabra = new PalabraOriginalBean();
        palabra.setId(Integer.parseInt(st2.nextToken()));
        palabra.setForm(st2.nextToken());
        palabra.setLemma(st2.nextToken());
        palabra.setUpos(st2.nextToken());
        palabra.setXpos(st2.nextToken());
        palabra.setFeats(st2.nextToken());
        palabra.setHead(Integer.parseInt(st2.nextToken()));
        palabra.setDeprel(st2.nextToken());
        palabra.setDeps(st2.nextToken());
        palabra.setMisc(st2.nextToken());
        lista.add(palabra);
    }
    return lista;
}
```

Figura 34. getWordsFromPhrase - UDPipeParser.java

4.4.6 Segmentador.java

Esta clase guarda el texto introducido en el sistema para analizar. Además, contiene un método que segmenta dicho texto en oraciones individuales y devuelve la lista con esas oraciones.

4.4.7 URLTools.java

Como ya hemos comentado anteriormente, nuestro sistema permite introducir una URL y la aplicación extraerá el texto de la página en cuestión para analizarlo. Para ello, hemos añadido al proyecto tres librerías externas: *boilerpipe-1.2.0.jar*, *nekohtml-1.9.18.jar* y *xerces-2.0.2.jar*. Dichas librerías nos proporcionarán clases y métodos que nos permitirán extraer textos de las páginas web para poder analizarlos en nuestra aplicación. En concreto, nuestra clase *URLTools.java* contendrá las funciones que utilizaremos para ello. Esta clase tiene dos métodos:

- *isURL ( )*, este método comprueba si lo que hemos introducido en el sistema es una URL o, por el contrario, se trata de un texto plano. Para ello, declaramos una expresión regular con la que tendremos que comparar el texto que hemos introducido. La expresión regular es la siguiente: **^(https?://)?(([\\w!~*'().&=+$%-]+:)?[\\w!~*'().&=+$%-]+@)?(([0-9]{1,3}\\.){3}[0-9]{1,3}|([\\w!~*'()-]+\\.)*([\\w^-][\\w-]{0,61})?[\\w]\\.[a-z]{2,6})(:[0-9]{1,4})?((/*)|(/+[\\w!~*'().;?:@&=+$,%#-]+)+/*)$.** Para comprobar si coincide, usaremos las clases *Pattern* y *Matcher*.
- *extractFromURL ( )*, este método extrae el texto de la página web que indique la URL. Para ello, usaremos clases de las librerías que hemos añadido al proyecto. Se trata de las clases *HTMLDocument*, *TextDocument*, *HTMLFetcher*, *BoilerpipeSAXInput*, *CommonExtractors*.

4.4.8 Graphviz.java

Esta clase implementa el código de todo lo que explicamos en el apartado 3.3.3. Su finalidad es imprimir los grafos de las oraciones que el sistema analice. Para ello, hemos creado cuatro métodos que detallamos a continuación:

- *getGraphString*: este método recibe tres listas: una con los *forms* de las palabras de la frase, otra con los *heads* y otra con los *deprels*. Estos tres campos del formato CoNLL-u son los que aparecerán en el grafo. Usando esas tres listas, crearemos una cadena de caracteres que contenga todo el contenido del fichero de texto que necesitamos pasarle al Graphviz. Devolveremos esa cadena.

```
public static String getGraphString(
        List<String> forms, List<Integer> heads, List<String> deprels) {
    StringBuilder graph = new StringBuilder();
    graph.append("digraph G" + '\n').append("{" + '\n');

    for (String form : forms) {
        graph.append('\t').append("\"").append(form).append("\"").append(";" + '\n');
    }
    graph.append('\t').append("ROOT").append(";" + '\n');
    graph.append('\n');
    for (int i = 0; i < forms.size(); i++) {
        if (heads.get(i).equals(0)) {
            graph.append('\t').append("\"").append(forms.get(i)).append("\"").append(" -> ")
                    .append("ROOT").append(" [label=")
                    .append("\"").append(deprels.get(i)).append("\"").append("];").append('\n');
        } else {
            graph.append('\t').append("\"").append(forms.get(i)).append("\"").append(" -> ")
                    .append("\"").append(forms.get(heads.get(i) - 1))
                    .append("\"").append(" [label=")
                    .append("\"").append(deprels.get(i)).append("\"").append("];").append('\n');
        }
    }
    graph.append("}");
    return graph.toString();
}
```

Figura 35. getGraphString - Graphviz.java

- *printGraphinFile*: este método es bastante simple. Creará un fichero de texto en el que escribirá toda la cadena de caracteres que hemos recibido del anterior método.

```
public static void printGraphInFile(String graph, int g, ServletContext context) {

    try {
        BufferedWriter writer
                = new BufferedWriter(new FileWriter("/temp" + "\\graph" + g + ".txt"));
        writer.write(graph);
        writer.close();
    } catch (IOException ex) {
        Logger.getLogger(Segmentador.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figura 36. printGraphInFile - Graphviz.java

- *getJPGGraph*: este es el método que crea la imagen con el grafo que la aplicación mostrará al analizar una frase. Para ello, y tal como explicamos en el apartado 3.3.3, necesitamos ejecutar un comando en la consola. Este método se encargará de ello utilizando el fichero de texto que creamos con el método anterior. Si todo va bien, se

creará un fichero .jpg con el grafo dibujado. Devolveremos el nombre de dicha imagen.

- *deleteFile*: este método solamente sirve para borrar los ficheros que hayamos creado.

4.4.9 ParserDB.java

Esta clase es la que se encarga del manejo de la base de datos de la aplicación. Contiene, por tanto, los métodos típicos de una clase de ese tipo: conexión y desconexión, inserción en las tablas, selección de registros, actualizar registros y limpiar la base de datos. La base de datos se crea con usuario "tfg" y contraseña "tfg". Vamos a profundizar un poco en los métodos más destacados:

- *createTables*: este método crea las tablas de la base de datos. Además, también añade al usuario **admin** a la tabla de usuarios.
- *limpiarBD*: este método destruye las cuatro tablas de la base de datos. Al final del mismo, se llama a *createTables* para volverlas a crear, de manera que nos queda la base de datos limpia (solo con el usuario **admin** en la tabla de usuarios).
  Debido a las referencias entre tablas, hay que borrarlas en un cierto orden: primero TABLA_MODIFICADA, después TABLA_ORIGINAL y por último FRASE_ORIGINAL. La tabla USUARIOS, al no referenciar ninguna de las otras tablas, puede borrar cuando queramos. En nuestro caso, la borramos en primer lugar.
- *authenticateUser:* este método recibe un *UserBean* de la página de login y busca en la base de datos si el usuario existe y si la contraseña es correcta.

```
public static String authenticateUser(UserBean ub) {
    String userNameDB = "", passwordDB = "", roleDB = "";
    String query = "select username,password,role from usuarios";
    try {
        pstmt = conn.prepareStatement(query);
        ResultSet result = pstmt.executeQuery();
        while (result.next()) {
            userNameDB = result.getString("username");
            passwordDB = result.getString("password");
            roleDB = result.getString("role");

            if (ub.getUserName().equals(userNameDB) &&
                    ub.getPassword().equals(passwordDB) && roleDB.equals("Admin")) {
                return "Admin_Role";
            } else if (ub.getUserName().equals(userNameDB) &&
                    ub.getPassword().equals(passwordDB) && roleDB.equals("User")) {
                return "User_Role";
            }
        }
        pstmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return "Error de login";
}
```

Figura 37. authenticateUser - ParserDB.java

El resto de métodos son los básicos para insertar, actualizar y seleccionar campos de la base de datos.

5 Integración, pruebas y resultados

5.1 Análisis de oración

Vamos a probar a analizar la misma oración, primero con UDPipe y luego con SpaCy, para poder comparar los resultados obtenidos. La oración que vamos a analizar es la siguiente: *Tu padre ha traído esta cartera para ti*.

En primer lugar, vamos a analizar la oración con UDPipe. El grafo resultante es el siguiente:

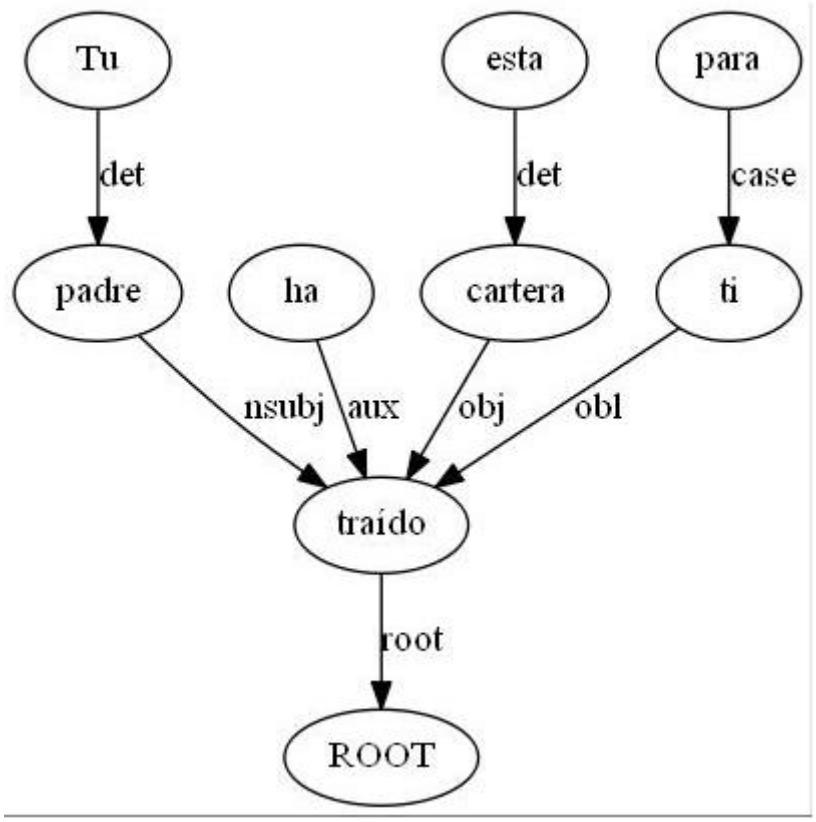


Figura 38. Prueba 1 – UDPipe

Después, vamos a proceder a analizar la oración con SpaCy. En la siguiente imagen se muestra el grafo resultante:

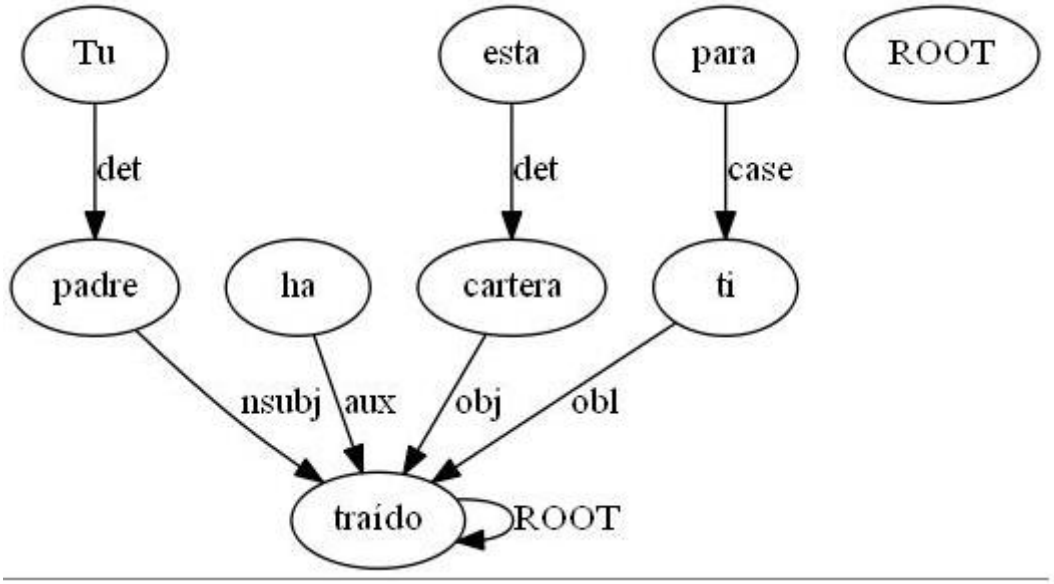


Figura 39. Prueba 1 – SpaCy

Como podemos comprobar, los dos grafos son idénticos. Esto nos indica que UDPipe y SpaCy propocionan aproximadamente el mismo nivel de precisión en sus análisis. Haremos otra prueba para cerciorarnos. Utilizaremos la siguiente oración: *El veloz zorro marrón salta sobre el perro vago.*

Al igual que en el caso anterior, analizaremos primero con UDPipe. Como resultado obtendremos el siguiente grafo:

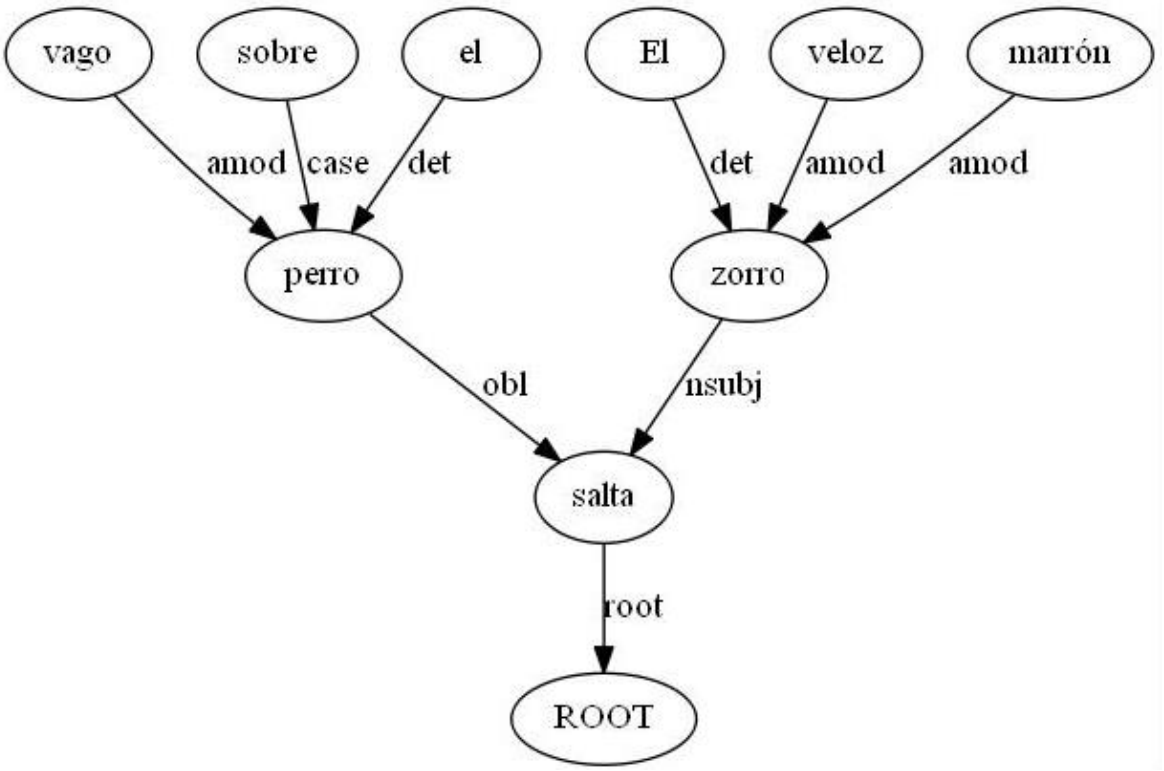


Figura 40 . Prueba 2 – UDPipe

Vamos ahora a analizar la misma oración con SpaCy:

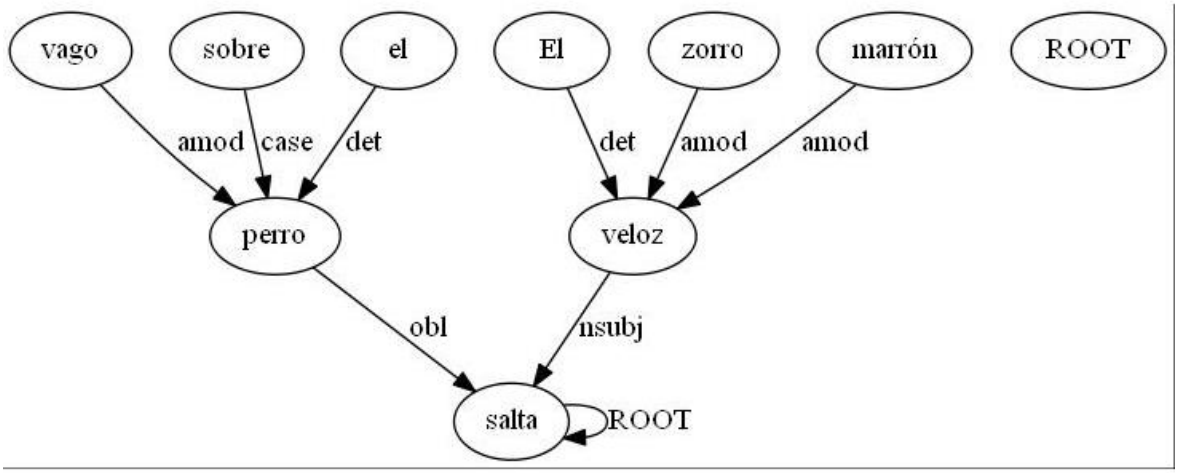


Figura 41. Prueba 2- SPaCy

Podemos ver que, al igual que en el caso anterior, los dos grafos salen iguales.
Por tanto, podemos afirmar que no hay un analizador mucho mejor que el otro, ya que ambos ofrecen un grado de precisión parecido.

5.2 Introducción de URL en el analizador

Vamos a probar una de las funciones del analizador: introducir una URL y analizar el contenido de la página web. En este caso, hemos introducido la URL del portal de Moodle de la Universidad: https://moodle.uam.es/. Hemos utilizado UDPipe para analizar las oraciones que la aplicación encuentre en la página. Vamos a mostrar los grafos de algunas de ellas.

- **Frase 1:** *¿Ha extraviado la contraseña?*

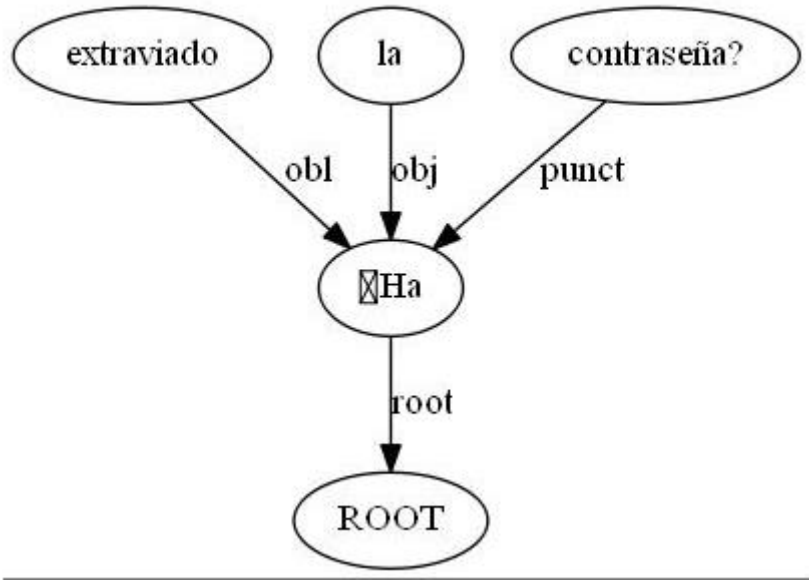


Figura 42. Frase 1 - Prueba URL - UDPipe

- **Frase 2:** *Ya está disponible la nueva herramienta antiplagio para Moodle, TURNITIN.*

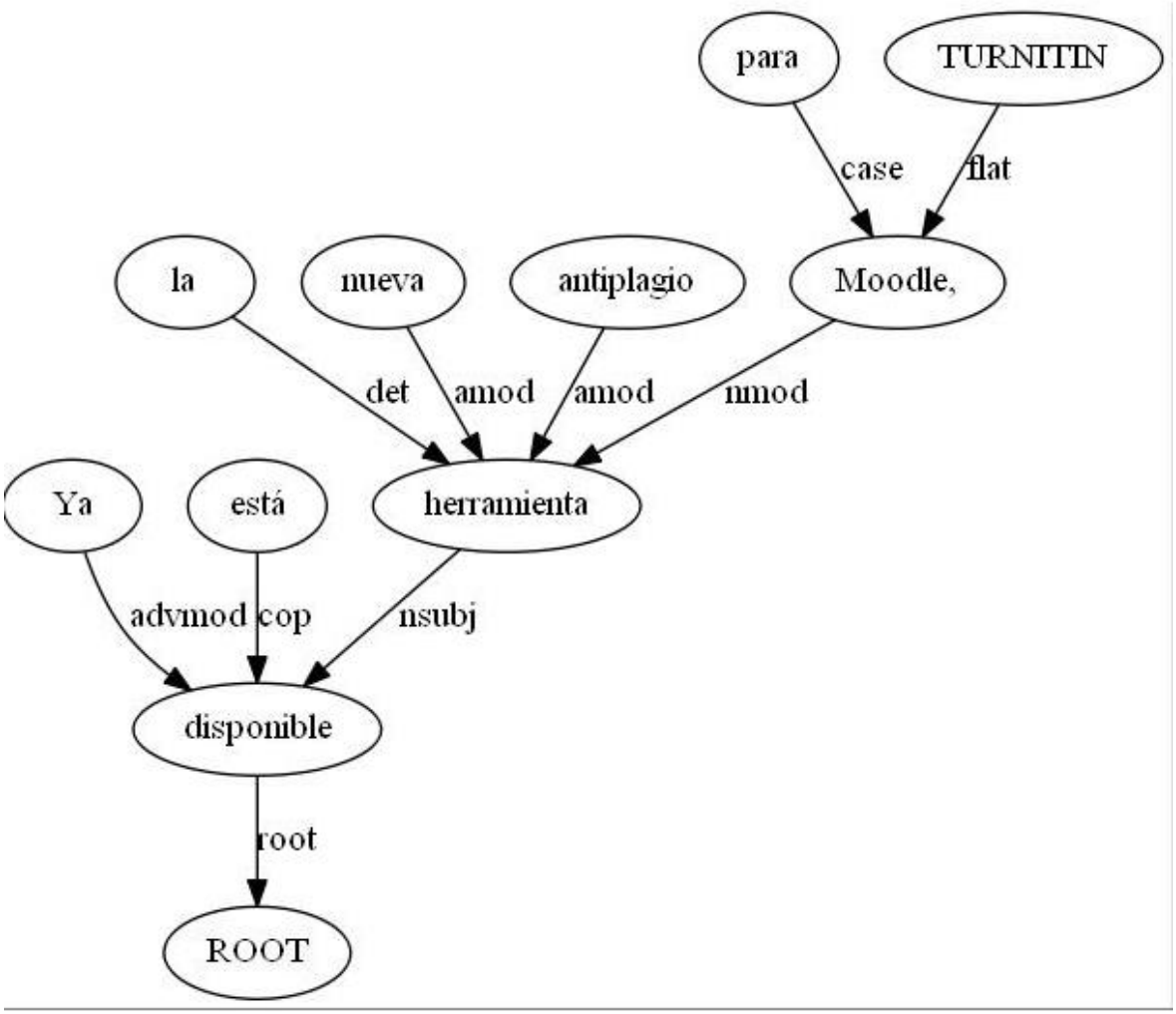


Figura 43. Frase 2 - Prueba URL - UDPipe

- **Frase 3:** *También será posible continuar utilizando la herramienta TURNITIN para evaluar trabajos de investigación.*

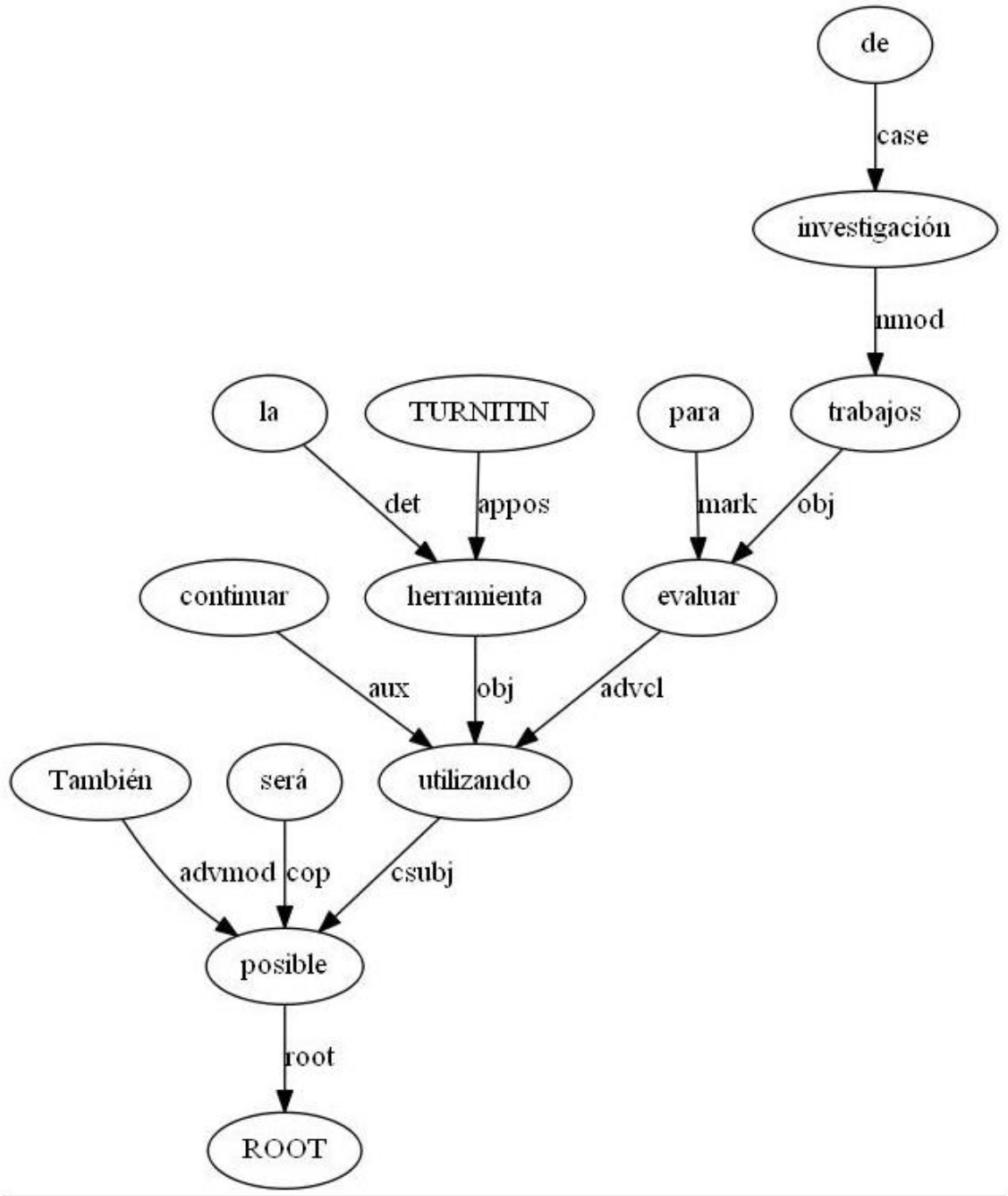


Figura 44. Frase 3 - Prueba URL - UDPipe

- **Frase 4:** *La ordenación docente correspondiente al curso académico 2018/19 está disponible en Moodle desde el 30 de julio, por lo que el profesorado ya puede acceder a los nuevos cursos cuya docencia tenga asignada en Sigma.*

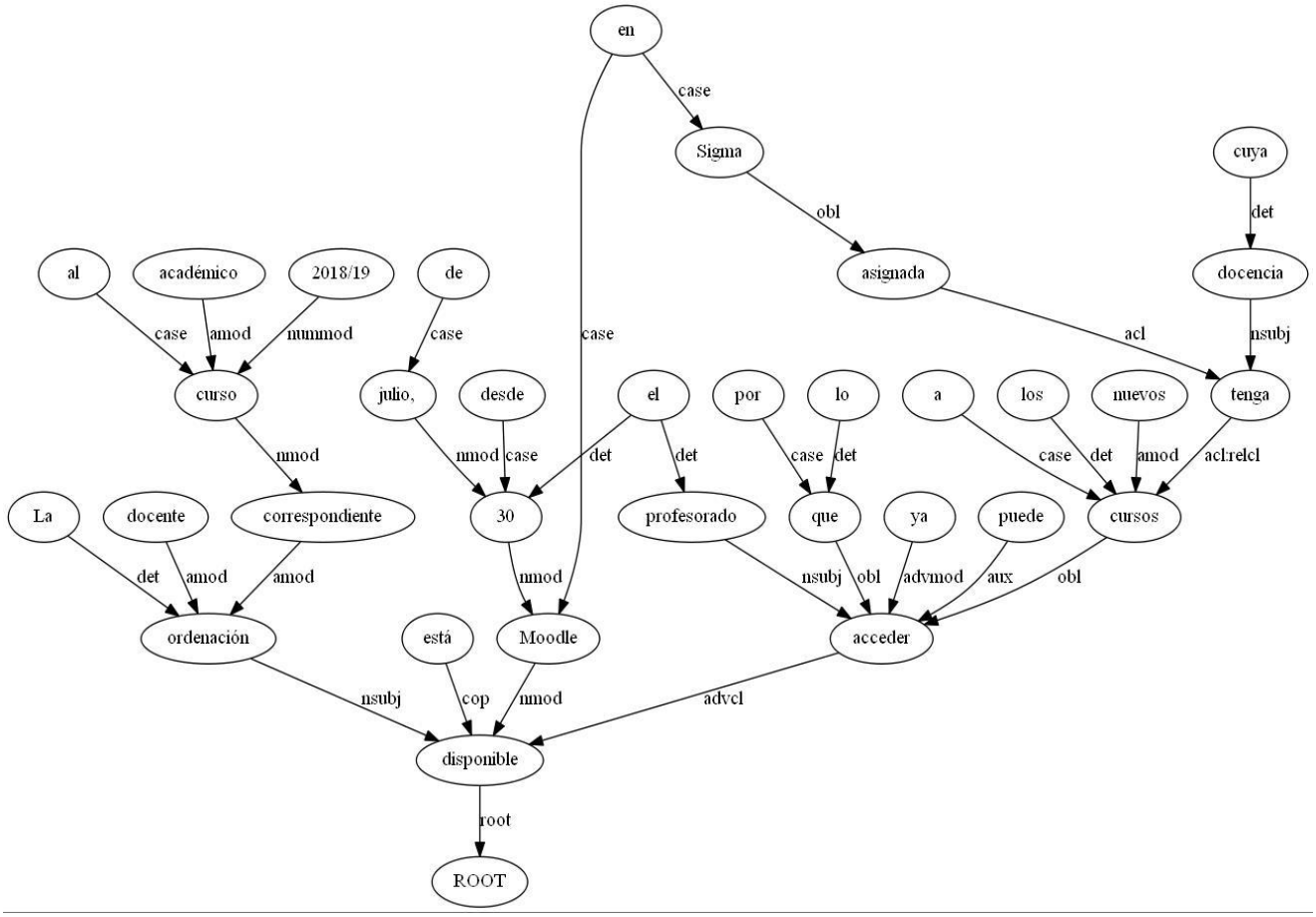


Figura 45. Frase 4 - Prueba URL - UDPipe

- **Frase 5:** *Si no aparece en Moodle toda su docencia, confirme en Sigma la asignación docente y, en caso de error, notifíquelo a la gestora de su Departamento.*

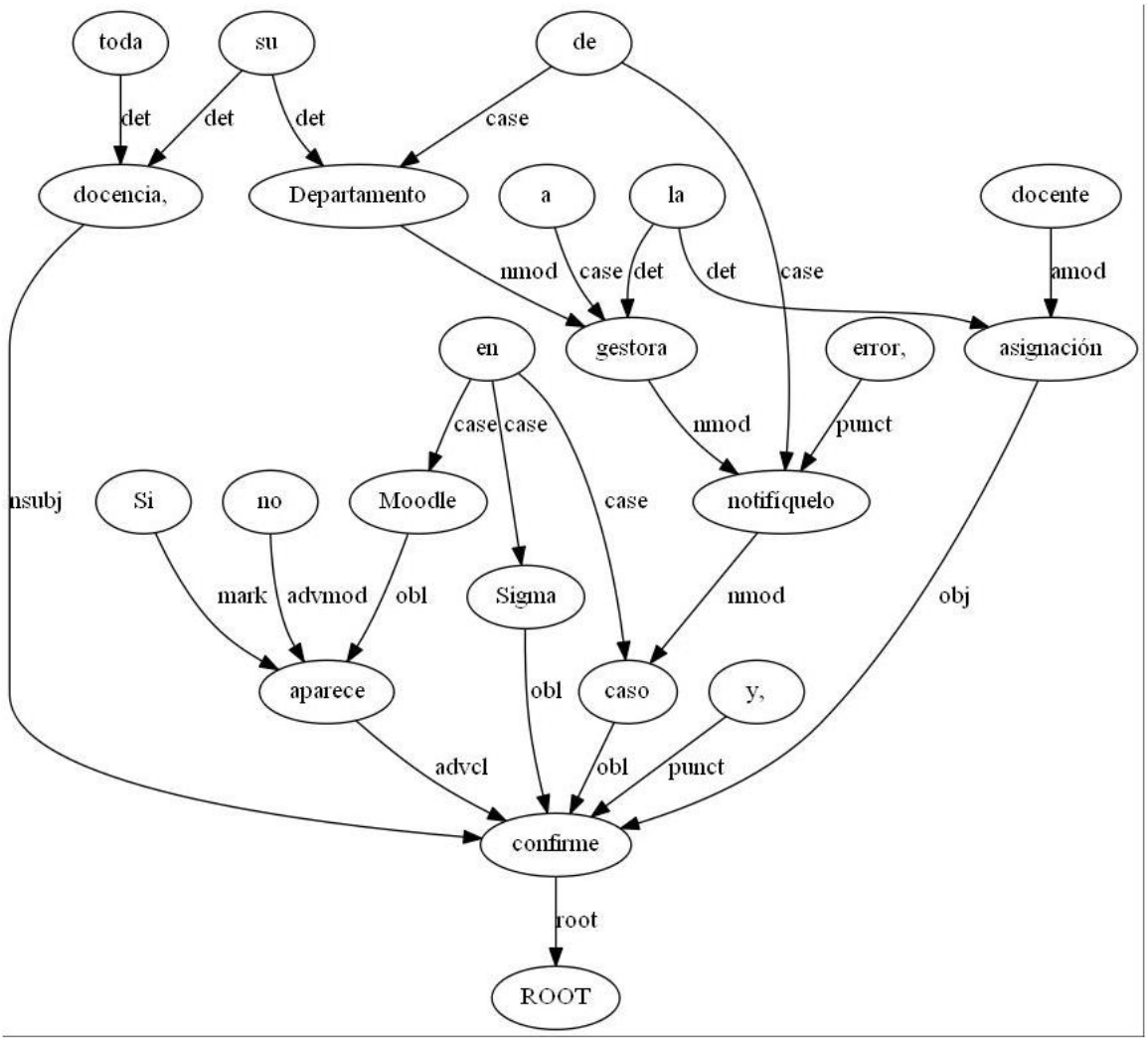


Figura 46. Frase 5 - Prueba URL – UDPipe

Podemos ver que la herramienta funciona correctamente. Vamos a analizar un par de frases más utilizando SpaCy. En este caso las frases serán de web http://uam.es/UAM/Home.htm?language=es.

- **Frase 1:** *Esta web utiliza cookies analíticas de terceros para facilitar la navegación y obtener información de uso de sus visitantes*

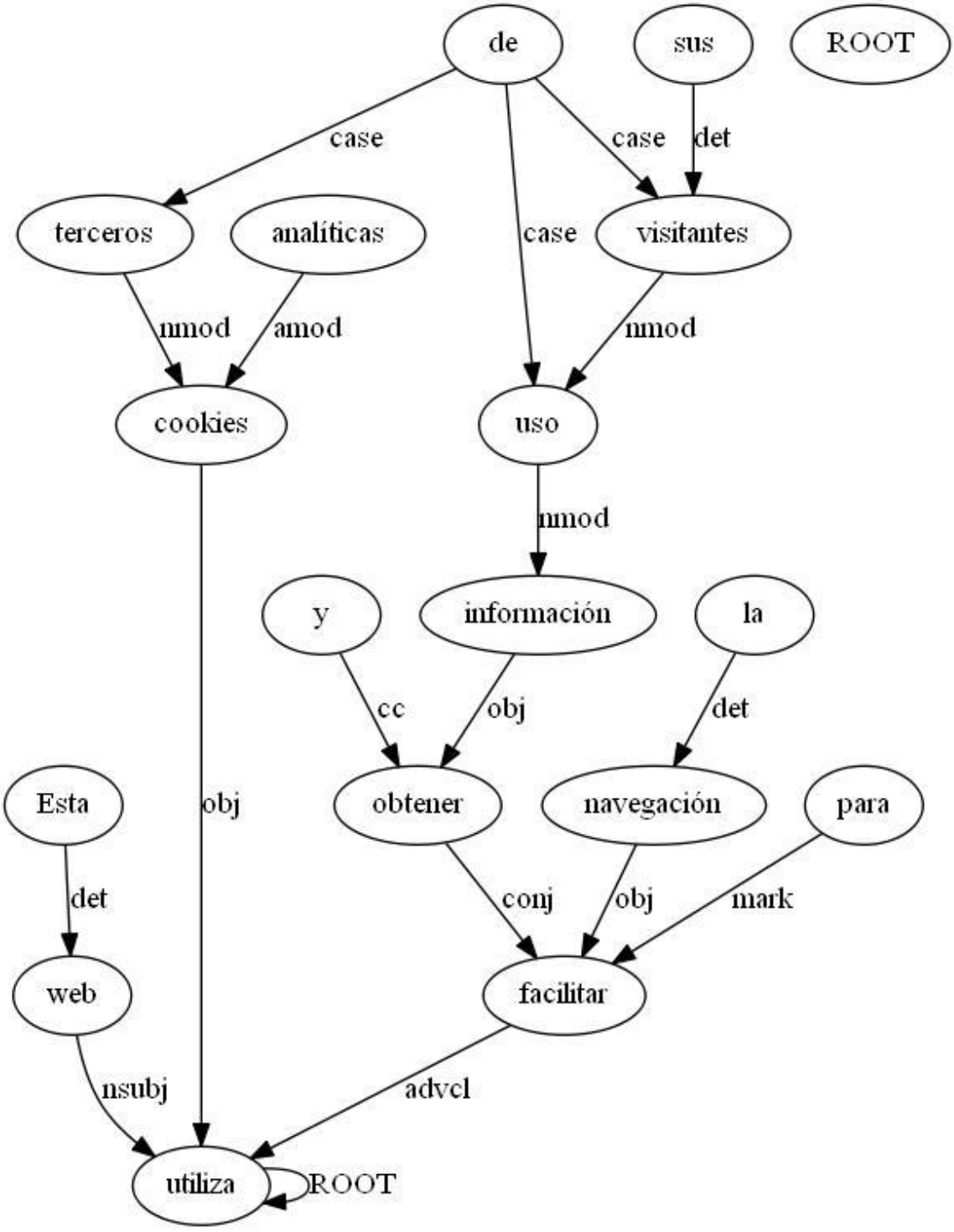


Figura 47. Frase 1 - Prueba URL – SpaCy

- **Frase 2:** *Puede obtener más información en nuestra política de cookies*

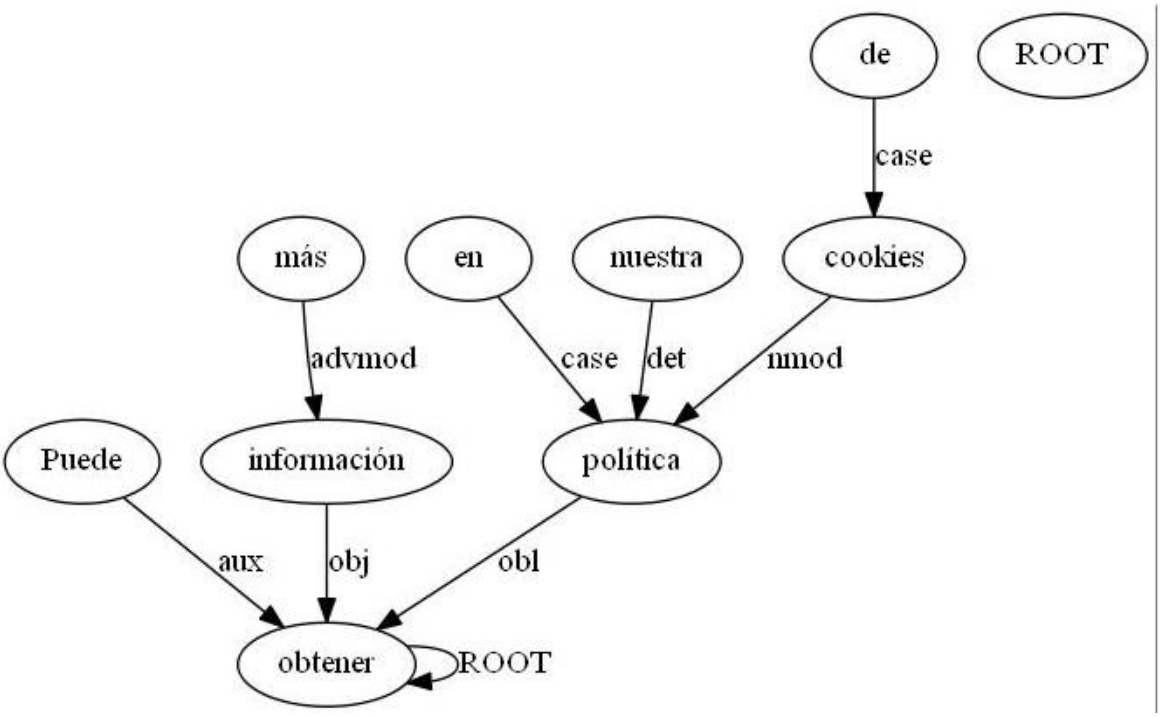


Figura 48. Frase 2 - Prueba URL – SpaCy

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Durante la realización de este Trabajo de Fin de Grado, he tenido la ocasión de incidir en ciertos aspectos que ya había visto sin tanta profundidad durante el transcurso de los estudios del grado.

También me ha permitido trabajar con herramientas y librerías que no conocía, y sobretodo ampliar mis conocimientos en el campo de la creación de servicios web. En este último aspecto hay que destacar especialmente el trabajo con las JSP y como se comunican usando los servlets. Ha resultado interesante además conocer herramientas como *Graphviz* y el trabajo con la base de datos.

Realizar este trabajo me ha permitido también investigar en otros campos fuera de la informática, como es la lingüística y, más concretamente, las gramáticas de dependencias. He conocido nuevos conceptos como el formato CoNLL-U y las herramientas que hay en línea para realizar los análisis sintácticos en los que se ha basado la aplicación creada.

Dicha aplicación se ha implementado de forma que pueda ser manejada intuitivamente por cualquier usuario.

En resumen, este trabajo ha resultado muy gratificante ya que me ha dado la oportunidad de aumentar mis conocimientos, tanto informáticos como lingüísticos.

6.2 Trabajo futuro

Como trabajo futuro, podrían implementarse algunas mejoras sobre el código o realizar una interfaz de usuario más atractiva para el usuario.

Podría incluirse una mejora que permitiera a la aplicación “entrenarse” en función de los resultados que va obteniendo y de las correcciones que se realizan por parte de los lingüistas.

Por otro lado, podría ser interesante mejorar la tecnología que se ha usado para construir la aplicación para hacerla más rápida y moderna, ya que los JSP/Servlets, aunque han resultado de utilidad y han cumplido con su deber, se han quedado algo anticuados y actualmente hay tecnologías más modernas y de mayor rendimiento. Este cometido requeriría un trabajo de reingeniería completa de la aplicación, por tanto, resultaría un trabajo arduo.

Referencias

[1] Chowdhury, G. (2003), Natural language processing. Annual Review of Information Science and Technology, 37. pp. 51-89. ISSN 0066-4200, http://dx.doi.org/10.1002/aris.1440370103

[2] F. S. Liddy (1999), Natural language processing. The journal of department of information science, Page 1-20.

[3] Y. D. González, Y. F. Romero (2003), Revista Telem@tica Vol. 11. Nº1, enero-abril, 2012, p. 47-57.

[4] Lucien Tesnière (1934), Comment construire une syntaxe.

[5] Fernando Carranza (2016), Tesnière y su Gramática de Dependencias: continuidades y discontinuidades. Revista argentina de historiografía lingüística, VIII, 2, 59-78, 2016.

[6] Milan Straka and Jana Strakova (2017), Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. Vancouver, Canadá, p. 88-99, http://www.aclweb.org/anthology/K/K17/K17-3009.pdf

[7] https://linguakit.com/es/sobre-linguakit

[8] https://universaldependencies.org/format.html

Glosario

API	*Application Programming Interface.*
Bootstrap	Framework para la creación de aplicaciones web.
JSP	*Java Server Pages*. Tecnología usada para desarrollar páginas web con HTML, XML y código Java embebido.
HTML	*HyperText Markup Language*.
XML	*Extensible Markup Language.*
Java	Lenguaje de programación orientado a objetos
Treebank	Texto cuyas oraciones están en forma de árbol.
Parser	Analizador
Derby	Gestor de bases de datos implementado en *Java*. Proporciona un driver *JDBC*.
Glassfish	Servidor de aplicaciones web perteneciente a *Oracle*.

Anexos

A Manual de instalación

La aplicación estará disponible en el siguiente repositorio de GitHub: https://github.com/jajuro/TFG_JavierJuarez. Solamente habrá que pulsar en el botón "Clone or Download". Después solamente habrá que pulsar en "Download ZIP" y, una vez descargado el fichero .zip, extraer de él el proyecto de NetBeans. Como podremos ver, también se extraerá el fichero **TFG.war**.

Para ejecutar nuestra aplicación tendremos que:

- Declarar la siguiente variable de entorno usando este comando en el cmd:
  setx GLASSFISH_HOME C:\glassfish-4.1.1 (o donde esté instalado Glassfish)
- Arrancar nuestro servidor. En nuestro caso, hemos usado Glassfish. Por tanto, para arrancarlo hemos usado el siguiente comando:
  C:\glassfish-4.1.1\glassfish\bin\asadmin start-domain
- Arrancar la base de datos Derby, usando el siguiente comando:
- ***C:\glassfish-4.1.1\glassfish\bin\asadmin start-database***
- Desplegar la aplicación. Para ello, una vez que el servidor está levantado, usaremos el siguiente comando:
  C:\glassfish-4.1.1\glassfish\bin\asadmin deploy <ruta del .war>\TFG.war
- Si no hay errores, entraremos en un navegador y escribiremos en la barra de direcciones la siguiente dirección: *http://localhost:8080/TFG*.
- En caso de que queramos parar el servidor y la base de datos, usaremos los siguientes comando:
  C:\glassfish-4.1.1\glassfish\bin\asadmin stop-database
  C:\glassfish-4.1.1\glassfish\bin\asadmin stop-domain

Hay que descargar algunas librerías externas que la aplicación utiliza. Son las siguientes:

- *udpipe.jar* (http://ufal.mff.cuni.cz/udpipe)
- *boilerpipe-1.2.0.jar* (http://www.java2s.com/Code/Jar/b/Downloadboilerpipe120jar.htm)
- *nekohtml-1.9.18.jar* (http://www.java2s.com/Code/Jar/n/Downloadnekohtml1918jar.htm)
- *xerces-2.0.2.jar* (http://www.java2s.com/Code/Jar/x/Downloadxerces202jar.htm)

Para que funcione correctamente, el usuario deberá tener instalado Python en su ordenador, de lo contrario el analizador *SpaCy* no funcionará.
Una vez esté Python instalado en nuestra máquina, procederemos a instalar SpaCy. Podemos instalarlo de 3 formas distintas:

- usando *pip*: lo haremos escribiendo el siguiente comando en la consola:
  pip install –U spacy (Windows, Linux y Mac)
- usando *conda*: en caso de usar *conda*, tendremos que escribir el siguiente comando:
  conda install –c conda-forge spacy (Windows, Linux y Mac)
- desde GitHub: para instalar SpaCy directamente desde el repositorio, habría que escribir lo siguiente:

Windows:
git clone https://github.com/explosion/spaCy
cd spaCy
set PYTHONPATH=/path/to/spaCy
pip install –r requirements.txt
python setup.py build_ext –inplace
Linux:
git clone https://github.com/explosion/spaCy
cd spaCy
export PYTHONPATH=`pwd`
pip install –r requirements.txt
python setup.py build_ext –inplace
Mac:
git clone https://github.com/explosion/spaCy
cd spaCy
pip install –r requirements.txt
python setup.py build_ext –inplace

En la web https://spacy.io/usage hay más detalles sobre como instalar SpaCy.

Una vez instalado, tendremos que instalar también los modelos que deseemos. En nuestro caso, nos interesante el modelo en español. Para instalarlo habría que escribir en la consola: ***python –m spacy download es***

Si tenemos la versión 2.7.9 o mayor de Python, o bien la 3.4 o mayor, *pip* vendrá instalado por defecto. Si no es así, para instalarlo habrá que hacer lo siguiente:

- Windows:
  - Descargar el fichero *get-pip.py* de la web https://bootstrap.pypa.io/get-pip.py
  - Entrar en la consola e ir al directorio donde hayamos descargado el fichero.
  - Ejecutar este comando: ***python get-pip.py***
- Linux:
  - Para Python 2.x, escribiremos en la consola: ***sudo apt-get install python-pip***
  - Para Python 3.x, escribiremos en la consola: ***sudo apt-get install python3-pip***
- Mac:
  - Usaremos el siguiente comando: ***sudo easy_install pip***

Para más información sobre la instalación de *pip*, se puede entrar en la web: https://www.makeuseof.com/tag/install-pip-for-python/

Para que el analizador UDPipe funcione, hay que meter los ficheros *udpipe_java.dll* y *udpipe.jar* en la carpeta *lib* del directorio del servidor de aplicaciones. Ambos ficheros se podrán obtener en la página de UDPipe: http://ufal.mff.cuni.cz/udpipe

Tendremos que descargar también Graphviz. Para ello iremos a la web https://graphviz.gitlab.io/_pages/Download/Download_windows.html y descargaremos el fichero graphviz-2.38.zip. Después lo extraeremos en nuestra máquina.